1. Dedication of "A First Course in Electrical and Computer Engineering"
2. Preface to "A First Course in Electrical and Computer Engineering"
3. Complex Numbers
    1. Complex Numbers: Introduction
    2. Complex Numbers: Geometry of Complex Numbers
    3. Complex Numbers: Algebra of Complex Numbers
    4. Complex Numbers: Roots of Quadratic Equations
    5. Complex Numbers: Representing Complex Numbers in a Vector Space
    6. Complex Numbers: An Electric Field Computation
    7. Complex Numbers: Numerical Experiment (Quadratic Roots)
4. The Functions e^x and e^jΘ
    1. The Functions e^x and e^jθ: Introduction
    2. The Functions e^x and e^jθ: The Function e^x
    3. The Functions e^x and e^jθ: The Function e^jθ and the Unit Circle
    4. The Functions e^x and e^jθ: The Euler and De Moivre Identities
    5. The Functions e^x and e^jθ: Roots of Unity and Related Topics
    6. The Functions e^x and e^jθ: Second-Order Differential and Difference Equations
    7. The Functions e^x and e^jθ: Numerical Experiment (Approximating e^jθ)
5. Phasors
    1. Phasors: Introduction
    2. Phasors: Phasor Representation of Signals
    3. Phasors: Beating between Tones

    4. Phasors: Multiphase Power
    5. Phasors: Lissajous Figures
    6. Phasors: Sinusoidal Steady State and the Series RLC Circuit
    7. Phasors: Light Scattering by a Slit
    8. Phasors: Numerical Experiment (Interference Patterns)
6. Linear Algebra
    1. Linear Algebra: Introduction
    2. Linear Algebra: Vectors
    3. Linear Algebra: Inner Product and Euclidean Norm
    4. Linear Algebra: Direction Cosines
    5. Linear Algebra: Projections
    6. Linear Algebra: Other Norms
    7. Linear Algebra: Matrices
    8. Linear Algebra: Solving Linear Systems of Equations
    9. Linear Algebra: Circuit Analysis
    10. Linear Algebra: Numerical Experiment (Circuit Design)
7. Vector Graphics
    1. Vector Graphics: Introduction
    2. Vector Graphics: Two-Dimensional Image Representation
    3. Vector Graphics: Two-Dimensional Image Transformations
    4. Vector Graphics: Composition of Transformations
    5. Vector Graphics: Homogeneous Coordinates
    6. Vector Graphics: Three-Dimensional Homogeneous Coordinates
    7. Vector Graphics: Projections
    8. Vector Graphics: Numerical Experiment (Star Field)
8. Filtering
    1. Filtering: Introduction
    2. Filtering: Simple Averages
    3. Filtering: Weighted Averages

    4. Filtering: Moving Averages
    5. Filtering: Exponential Averages and Recursive Filters
    6. Filtering: Test Sequences
    7. Filtering: Numerical Experiment (Frequency Response of First-Order Filter)
9. Binary Codes
    1. Binary Codes: Introduction
    2. Binary Codes: The Communication Paradigm
    3. Binary Codes: From Symbols to Binary Codes
    4. Binary Codes: Huffman Codes for Source Coding
    5. Binary Codes: Hamming Codes for Channel Coding
    6. Binary Codes: Numerical Experiment (Huffman Codes)
10. An Introduction to MATLAB
    1. An Introduction to MATLAB: Introduction
    2. An Introduction to MATLAB: Running MATLAB (Macintosh)
    3. An Introduction to MATLAB: Running MATLAB (PC)
    4. An Introduction to MATLAB: Interactive Mode
    5. An Introduction to MATLAB: Variables
    6. An Introduction to MATLAB: Complex Variables
    7. An Introduction to MATLAB: Vectors and Matrices
    8. An Introduction to MATLAB: The Colon
    9. An Introduction to MATLAB: Graphics
    10. An Introduction to MATLAB: Editing Files and Creating Functions (Macintosh)
    11. An Introduction to MATLAB: Editing Files and Creating Functions (PC)
    12. An Introduction to MATLAB: Loops and Control
11. The Edix Editor
12. Useful Mathematical Identities

Dedication of "A First Course in Electrical and Computer Engineering"

Louis Scharf dedicates this book to
his wife Carol, son Greg, and daughter Heidi,
for love and inspiration;
his parents Louis Sr. and Ann,
in celebration of their 50th wedding anniversary.

Richard Behrens dedicates this book to
his wife Debbie, and child as yet unborn,
for love and encouragement;
his parents Richard and Elsie,
in gratitude for a good education.

Preface to "A First Course in Electrical and Computer Engineering"

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

This book was written for an experimental freshman course at the University of Colorado. The course is now an elective that the majority of our electrical and computer engineering students take in the second semester of their freshman year, just before their first circuits course. Our department decided to offer this course for several reasons:

1. we wanted to pique student' interest in engineering by acquainting them with engineering teachers early in their university careers and by providing with exposure to the types of problems that electrical and computer engineers are asked to solve;
2. we wanted students entering the electrical and computer engineering programs to be prepared in complex analysis, phasors, and linear algebra, topics that are of fundamental importance in our discipline;
3. we wanted students to have an introduction to a software application tool, such as MATLAB, to complete their preparation for practical and efficient computing in their subsequent courses and in their professional careers;
4. we wanted students to make early contact with advanced topics like vector graphics, filtering, and binary coding so that they would gain a more rounded picture of modern electrical and computer engineering.

In order to introduce this course, we had to sacrifice a second semester of Pascal programming. We concluded that the sacrifice was worth making because we found that most of our students were prepared for high-level language computing after just one semester of programming.

We believe engineering educators elsewhere are reaching similar conclusions about their own students and curriculums. We hope this book helps create a much needed dialogue about curriculum revision and that it leads to the development of similar introductory courses that encourage students to enter and practice our craft.

Students electing to take this course have completed one semester of calculus, computer programming, chemistry, and humanities. Concurrently with this course, students take physics and a second semester of calculus, as well as a second semester in the humanities. By omitting the advanced topics marked by asterisks, we are able to cover Complex Numbers through Linear Algebra, plus two of the three remaining chapters. The book is organized so that the instructor can select any two of the three. If every chapter of this book is covered, including the advanced topics, then enough material exists for a two-semester course.

The first three chapters of this book provide a fairly complete coverage of complex numbers, the functions e^x and $e^{j\theta}$, and phasors. Our department philosophy is that these topics must be understood if a student is to succeed in electrical and computer engineering. These three chapters may also be used as a supplement to a circuits course. A measured pace of presentation, taking between sixteen and eighteen lectures, is sufficient to cover all but the advanced sections in Complex Numbers through Phasors.

The chapter on "linear algebra" is prerequisite for all subsequent chapters. We use eight to ten lectures to cover it. We devote twelve to sixteen lectures to cover topics from Vector Graphics through Binary Codes. (We assume a semester consisting of 42 lectures and three exams.) The chapter on vector graphics applies the linear algebra learned in the previous chapter to the problem of translating, scaling, and rotating images. "Filtering" introduces the student to basic ideas in averaging and filtering. The chapter on "Binary Codes" covers the rudiments of binary coding, including Huffman codes and Hamming codes.

If the users of this book find "Vector Graphics" through "Binary Codes" too confining, we encourage them to supplement the essential material in "Complex Numbers" through "Linear Algebra" with their own course notes on additional topics. Within electrical and computer engineering there are

endless possibilities. Practically any set of topics that can be taught with conviction and enthusiasm will whet the student's appetite. We encourage you to write to us or to our editor, Tom Robbins, about your ideas for additional topics. We would like to think that our book and its subsequent editions will have an open architecture that enables us to accommodate a wide range of student and faculty interests.

Throughout this book we have used MATLAB programs to illustrate key ideas. MATLAB is an interactive, matrix-oriented language that is ideally suited to circuit analysis, linear systems, control theory, communications, linear algebra, and numerical analysis. MATLAB is rapidly becoming a standard software tool in universities and engineering companies. (For more information about MATLAB, return the attached card in the back of this book to The MathWorks, Inc.) MATLAB programs are designed to develop the student's ability to solve meaningful problems, compute, and plot in a high-level applications language. Our students get started in MATLAB by working through "An Introduction to MATLAB," while seated at an IBM PC (or look-alike) or an Apple Macintosh. We also have them run through the demonstration programs in "Complex Numbers". Each week we give three classroom lectures and conduct a one-hour computer lab session. Students use this lab session to hone MATLAB skills, to write programs, or to conduct the numerical experiments that are given at the end of each chapter. We require that these experiments be carried out and then reported in a short lab report that contains (i) introduction, (ii) analytical computations, (iii) computer code, (iv) experimental results, and (v) conclusions. The quality of the numerical results and the computer graphics astonishes students. Solutions to the chapter problems are available from the publisher for instructors who adopt this text for classroom use.

We wish to acknowledge our late colleague Richard Roberts, who encouraged us to publish this book, and Michael Lightner and Ruth Ravenel, who taught "Linear Algebra" and "Vector Graphics" and offered helpful suggestions on the manuscript. We thank C. T. Mullis for allowing us to use his notes on binary codes to guide our writing of "Binary Codes". We thank Cédric Demeure and Peter Massey for their contributions to the writing of "An Introduction to MATLAB" and "The Edix Editor". We thank Tom Robbins, our editor at Addison-Wesley, for his encouragement,

patience, and many suggestions. We are especially grateful to Julie Fredlund, who composed this text through many drafts and improved it in many ways. We thank her for preparing an excellent manuscript for production.

L. L. Scharf
R. T. Behrens
Boulder, Colorado

To the Teacher:

An incomplete understanding of complex numbers and phasors handicaps students in circuits and electronics courses, and even more so in advanced courses such as electromagnetics. optics, linear systems, control, and communication systems. Our faculty has decided to address this problem as early as possible in the curriculum by designing a course that drills complex numbers and phasors into the minds of beginning engineering students. We have used power signals, musical tones, Lissajous figures, light scattering, and *RLC* circuits to illustrate the usefulness of phasor calculus. "Linear Algebra" through "Binary Codes" introduce students to a handful of modern ideas in electrical and computer engineering. The motivation is to whet students' appetites for more advanced problems. The topics we have chosen – linear algebra, vector graphics, filtering, and binary codes – are only representative.

At first glance, many of the equations in this book look intimidating to beginning students. For this reason, we proceed at a very measured pace. In our lectures, we write out in agonizing detail every equation that involves a sequence or series. For example, the sum $\sum_{n=0}^{N-1} z^n$ is written out as

Equation:

$$1 + z + z^2 + \cdots + z^{N-1},$$

and then it is evaluated for some specific value of *z* before we derive the analytical result $\frac{1-z^N}{1-z}$ Similarly, an infinite sequence like $\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n$ is written out as

Equation:

$$(1+x),\ \left(1+\frac{x}{2}\right)^2,\ \left(1+\frac{x}{3}\right)^3,\ ...,\ \left(1+\frac{x}{100}\right)^{100},\ ...,$$

and then it is evaluated for some specific *x* and for several values of *n* before the limit is derived. We try to preserve this practice of pedantic excess until it is clear that every student is comfortable with an idea and the notation for coding the idea.

To the Student:

These are exciting times for electrical and computer engineering. To celebrate its silver anniversary, the National Academy of Engineering announced in February of 1990 the top ten engineering feats of the previous twenty-five years. The Apollo moon landing, a truly Olympian and protean achievement, ranked number one. However, a number of other achievements in the top ten were also readily identifiable as the products of electrical and computer engineers:

1. communication and remote sensing satellites,
2. the microprocessor,
3. computer-aided design and manufacturing (CADCAM),
4. computerized axial tomography (CAT scan),
5. lasers, and
6. fiber optic communication.

As engineering students, you recognize these achievements to be important milestones for humanity; you take pride in the role that engineers have played in the technological revolution of the twentieth century.

So how do we harness your enthusiasm for the grand enterprise of engineering? Historically, we have enrolled you in a freshman curriculum of mathematics, science, and humanities. If you succeeded, we enrolled you in an engineering curriculum. We then taught you the details of your profession and encouraged your faith that what you were studying is what you must study to be creative and productive engineers. The longer your faith held, the more likely you were to complete your studies. This seems like an imperious approach to engineering education, even though mathematics, physics, and the humanities are the foundation of engineering, and details are what form the structure of engineering. It seems to us that a better way to stimulate your enthusiasm and encourage your faith is to introduce you early in your studies to engineering teachers who will share their insights about some of the fascinating advanced topics in engineering, while teaching you the mathematical and physical principles of engineering. But you must match the teacher's commitment with your own commitment to study. This means that you must attend lectures, read texts, and work problems. You must be inquisitive and skeptical. Ask yourself how an idea is limited in scope and how it might be extended to apply to a wider range of problems. For, after all, one of the great themes of engineering is that a few fundamental ideas from mathematics and science, coupled with a few principles of design, may be applied to a wide range of engineering problems. Good luck with your studies.

Complex Numbers: Introduction

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Notes to Teachers and Students:

When we teach complex numbers to beginning engineering students, we encourage a *geometrical picture* supported by an *algebraic structure*. Every algebraic manipulation carried out in a lecture is accompanied by a carefully drawn picture in order to fix the idea that geometry and algebra go hand-in-glove to complete our understanding of complex numbers. We assign essentially every problem for homework.

We use the MATLAB programs in this chapter to illustrate the theory of complex numbers and to develop skill with the MATLAB language. The numerical experiment introduces students to *the* basic quadratic equation of electrical and computer engineering and shows how the roots of this quadratic equation depend on the coefficients of the equation.

"Representing Complex Numbers in a Vector Space," is a little demanding for freshmen but easily accessible to sophomores. It may be covered for additional insight, skipped without consequence, or covered after Chapter 4. "An Electric Field Computation," is well beyond most freshmen, and it is demanding for sophomores. Nonetheless, an expert in electromagnetics might want to cover the section "An electric Field Computation" for the insight it brings to the use of complex numbers for representing two-dimensional real quantities.

Introduction

It is hard to overestimate the value of complex numbers. They first arose in the study of roots for quadratic equations. But, as with so many other great discoveries, complex numbers have found widespread application well outside their original domain of discovery. They are now used throughout mathematics, applied science, and engineering to represent the harmonic nature of vibrating systems and oscillating fields. For example, complex numbers may be used to study

i. traveling waves on a sea surface;
ii. standing waves on a violin string;
iii. the pure tone of a Kurzweil piano;
iv. the acoustic field in a concert hall;
v. the light of a He-Ne laser;
vi. the electromagnetic field in a light show;
vii. the vibrations in a robot arm;
viii. the oscillations of a suspension system;
ix. the carrier signal used to transmit AM or FM radio;
x. the carrier signal used to transmit digital data over telephone lines; and
xi. the 60 Hz signal used to deliver power to a home.

In this chapter we develop the algebra and geometry of complex numbers. In Chapter 3 we will show how complex numbers are used to build phasor representations of power and communication signals.

Complex Numbers: Geometry of Complex Numbers

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

The most fundamental new idea in the study of complex numbers is the "imaginary number" j. This imaginary number is defined to be the square root of -1:
Equation:

$$j = \sqrt{-1}$$

Equation:

$$j^2 = -1.$$

The imaginary number j is used to build complex numbers x and y in the following way:
Equation:

$$z = x + jy.$$

We say that the complex number z has "real part" x and "imaginary part" y:
Equation:

$$z = \mathrm{Re}[z] + j\,\mathrm{Im}[z]$$

Equation:

$$\mathrm{Re}[z] = x;\ \ \mathrm{Im}[z] = y.$$

In MATLAB, the variable x is denoted by `real(z)`, and the variable y is denoted by `imag(z)`. In communication theory, x is called the "in-phase" component of z, and y is called the "quadrature" component. We call $z = x + jy$ the *Cartesian* representation of z, with real component x and imaginary component y. We say that the Cartesian pair (x, y)*codes* the complex number z.

We may plot the complex number z on the plane as in [link]. We call the horizontal axis the "real axis" and the vertical axis the "imaginary axis." The plane is called the "complex plane." The radius and angle of the line to the point $z = x + jy$ are

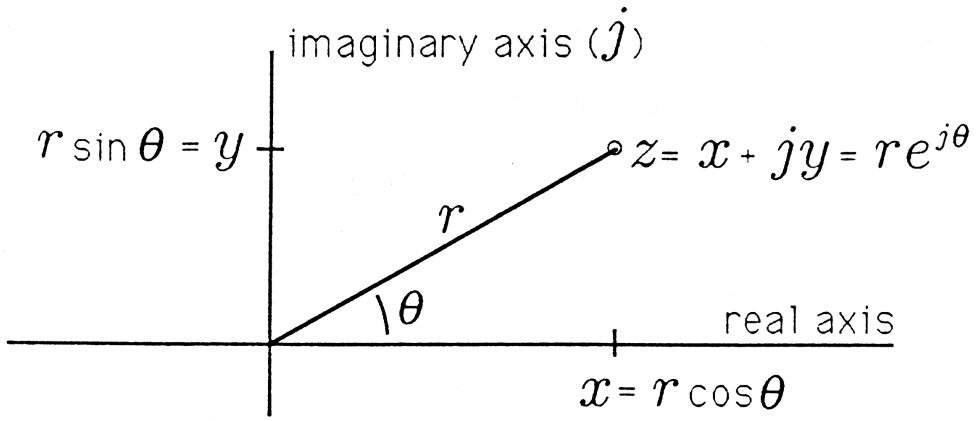
Equation:

$$r = \sqrt{x^2 + y^2}$$

Equation:

$$\theta = \tan^{-1}\left(\frac{y}{x}\right).$$

See [link]. In MATLAB, r is denoted by `abs(z)`, and θ is denoted by `angle(z)`.



Cartesian and Polar Representations of the Complex Number z

The original Cartesian representation is obtained from the radius r and angle θ as follows:
Equation:

$$x = r\cos\theta$$

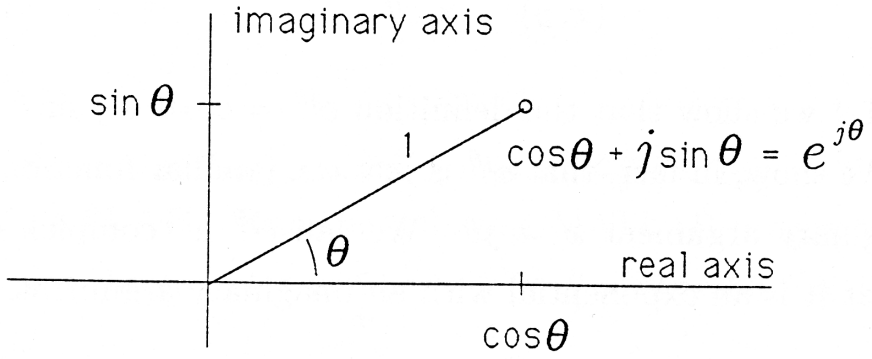
Equation:

$$y = r\ sin\ \theta.$$

The complex number z may therefore be written as
Equation:

$$\begin{aligned} z &= x + jy \\ &= r\cos\theta + jr\sin\theta \\ &= r\ (\cos\theta + j\,sin\ \theta). \end{aligned}$$

The complex number $\cos\theta + j\sin\theta$ is, itself, a number that may be represented on the complex plane and coded with the Cartesian pair $(\cos\theta, \sin\theta)$. This is illustrated in [link]. The radius and angle to the point $z = \cos\theta + j\sin\theta$ are 1 and θ. Can you see why?



The Complex Number $\cos\ \theta + j\ \sin\ \theta$

The complex number $\cos\theta + j\sin\theta$ is of such fundamental importance to our study of complex numbers that we give it the special symbol $e^{j\theta}$:
Equation:

$$e^{j\theta} = \cos\theta + j\sin\theta.$$

As illustrated in [link], the complex number $e^{j\theta}$ has radius 1 and angle θ. With the symbol $e^{j\theta}$, we may write the complex number z as
Equation:

$$z = re^{j\theta}.$$

We call $z = re^{j\theta}$ a *polar* representation for the complex number z. We say that the polar pair $r\angle\theta$*codes* the complex number z. In this polar representation, we define $|z| = r$ to be the *magnitude* of z and $\arg(z) = \theta$ to be the angle, or *phase*, of z:
Equation:

$$|z| = r$$

Equation:

$$arg\ (z) = \theta.$$

With these definitions of magnitude and phase, we can write the complex number z as
Equation:

$$z = |z|e^{j\arg(z)}.$$

Let's summarize our ways of writing the complex number *z* and record the corresponding geometric codes:
Equation:

$$\begin{array}{ccccccc} z & = & x+jy & = & re^{j\theta} & = & \left|z\right|e^{j\ \arg(z)}. \\ & & \downarrow & & \downarrow & & \\ & & (x,y) & & r\angle\theta & & \end{array}$$

In "Roots of Quadratic Equations" we show that the definition $e^{j\theta} = \cos\theta + j\sin\theta$ is more than symbolic. We show, in fact, that $e^{j\theta}$ is just the familiar function e^x evaluated at the imaginary argument $x = j\theta$. We call $e^{j\theta}$ a "complex exponential," meaning that it is an exponential with an imaginary argument.

Exercise:

Problem:

Prove $(j)^{2n} = (-1)^n$ and $(j)^{2n+1} = (-1)^n j$. Evaluate j^3, j^4, j^5.

Exercise:

Problem:

Prove $e^{j[(\pi/2)+m2\pi]} = j$, $e^{j[(3\pi/2)+m2\pi]} = -j$, $e^{j(0+m2\pi)} = 1$, and $e^{j(\pi+m2\pi)} = -1$. Plot these identities on the complex plane. (Assume m is an integer.)

Exercise:

Problem:

Find the polar representation $z = re^{j\theta}$ for each of the following complex numbers:

a. $z = 1 + j0$;
b. $z = 0 + j1$;
c. $z = 1 + j1$;
d. $z = -1 - j1$.

Plot the points on the complex plane.

Exercise:

Problem:

Find the Cartesian representation $z = x + jy$ for each of the following complex numbers:

a. $z = \sqrt{2}e^{j\pi/2}$;
b. $z = \sqrt{2}e^{j\pi/4}$;
c. $z = e^{j3\pi/4}$;
d. $z = \sqrt{2}e^{j3\pi/2}$.

Plot the points on the complex plane.

Exercise:

Problem:

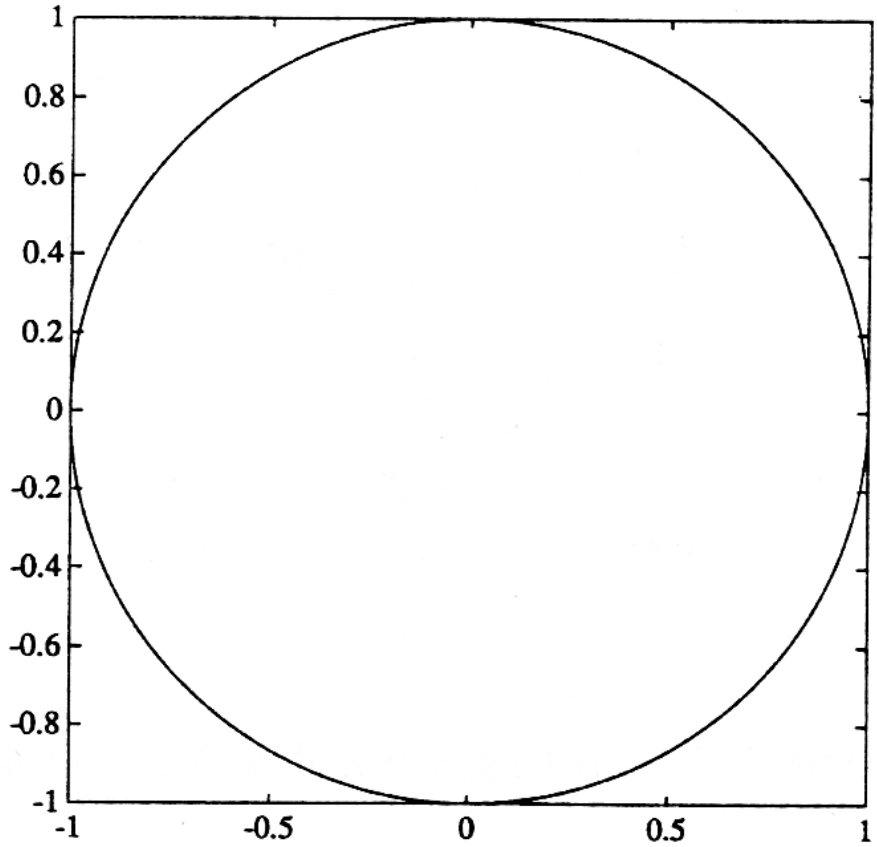
The following geometric codes represent complex numbers. Decode each by writing down the corresponding complex number *z*:

a. $(0.7, -0.1) \quad z = ?$
b. $(-1.0, 0.5) \quad z = ?$
c. $1.6\angle\pi/8 \quad z =?$
d. $0.4\angle 7\pi/8 \quad z =?$

Exercise:

Problem:

Show that $\mathrm{Im}[jz] = \mathrm{Re}[z]$ and $\mathrm{Re}[-jz] = \mathrm{Im}[z]$. Demo 1.1 (MATLAB). Run the following MATLAB program in order to compute and plot the complex number $e^{j\theta}$ for $\theta = i2\pi/360, i = 1, 2, ..., 360$: `j=sqrt(-1) n=360 for i=1:n,circle(i)=exp(j*2*pi*i/n);end; axis('square') plot(circle)` Replace the explicit for loop of line 3 by the implicit loop `circle=exp(j*2*pi*[1:n]/n);` to speed up the calculation. You can see from [link] that the complex number $e^{j\theta}$, evaluated at angles $\theta = 2\pi/360, 2(2\pi/360), ...,$ turns out complex numbers that lie at angle θ and radius 1. We say that $e^{j\theta}$ is a complex number that lies on the unit circle. We will have much more to say about the unit circle in Chapter 2.



The Complex Numbers $e^{j\theta}$ for $0 \leq \theta \leq 2\pi$ (Demo 1.1)

Complex Numbers: Algebra of Complex Numbers

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

The complex numbers form a mathematical “field” on which the usual operations of addition and multiplication are defined. Each of these operations has a simple geometric interpretation.

Addition and Multiplication.

The complex numbers z_1 and z_2 are

added according to the rule
Equation:

$$\begin{aligned} z_1 + z_2 &= (x_1 + jy_1) + (x_2 + jy_2) \\ &= (x_1 + x_2) + j(y_1 + y_2). \end{aligned}$$

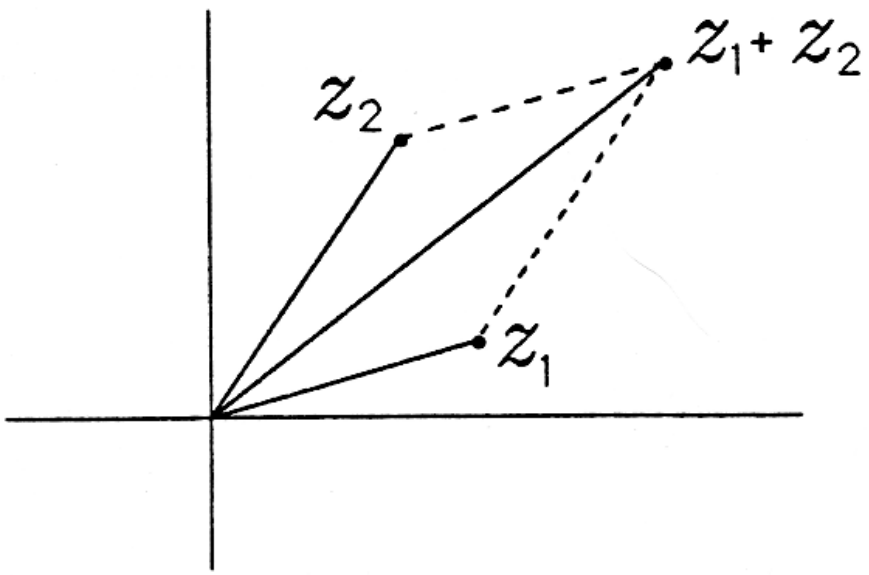
We say that the real parts add and the imaginary parts add. As illustrated in [link], the complex number $z_1 + z_2$ is computed from a “parallelogram rule,” wherein $z_1 + z_2$ lies on the node of a parallelogram formed from z_1 and z_2.
Exercise:

Problem:

Let $z_1 = r_1 e^{j\theta_1}$ and $z_2 = r_2 e^{j\theta_2}$. Find a polar formula $z_3 = r_3 e^{j\theta_3}$ for $z_3 = z_1 + z_2$ that involves *only* the variables r_1, r_2, θ_1, and θ_2. The formula for r_3 is the “law of cosines.”

The product of z_1 and z_2 is
Equation:

$$\begin{aligned} z_1 z_2 &= (x_1 + jy_1)(x_2 + jy_2) \\ &= (x_1 x_2 - y_1 y_2) + j(y_1 x_2 + x_1 y_2). \end{aligned}$$

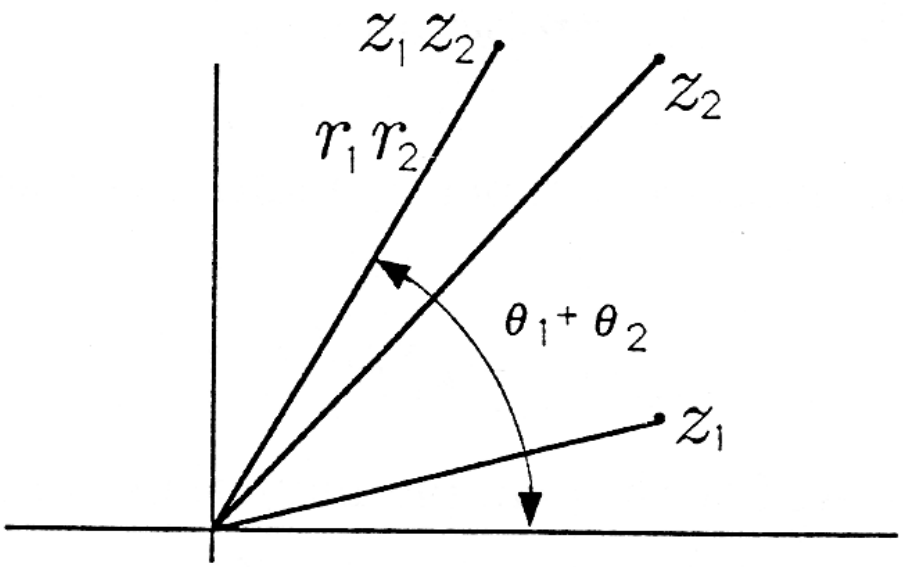


Adding Complex Numbers

If the polar representations for z_1 and z_2 are used, then the product may be written as [footnote]
We have used the trigonometric identities cos $(\theta_1 + \theta_2)$ =cos θ_1 cos θ_2− sin θ_1 sin θ_2 and sin $(\theta_1 + \theta_2)$ =sin θ_1 cos θ_2+ cos θ_1 sin θ_2 to derive this result.
Equation:

$$
\begin{aligned}
z_1 z_2 &= r_1 e^{j\theta_1} r_2 e^{j\theta_2} \\
&= (r_1 \cos\theta_1 + j r_1 \sin\theta_1)(r_2 \cos\theta_2 + j r_2 \sin\theta_2) \\
&= (\, r_1 \, cos \, \theta_1 r_2 \, cos \, \theta_2 - r_1 \, sin \, \theta_1 r_2 \, sin \, \theta_2) \\
&+ \; j(\, r_1 \, sin \, \theta_1 r_2 \, cos \, \theta_2 + r_1 \, cos \, \theta_1 r_2 \, sin \, \theta_2) \\
&= r_1 r_2 \cos(\theta_1 + \theta_2) + j r_1 r_2 \sin(\theta_1 + \theta_2) \\
&= r_1 r_2 e^{j(\theta_1 + \theta_2)}.
\end{aligned}
$$

We say that the magnitudes multiply and the angles add. As illustrated in [link], the product $z_1 z_2$ lies at the angle $(\theta_1 + \theta_2)$.



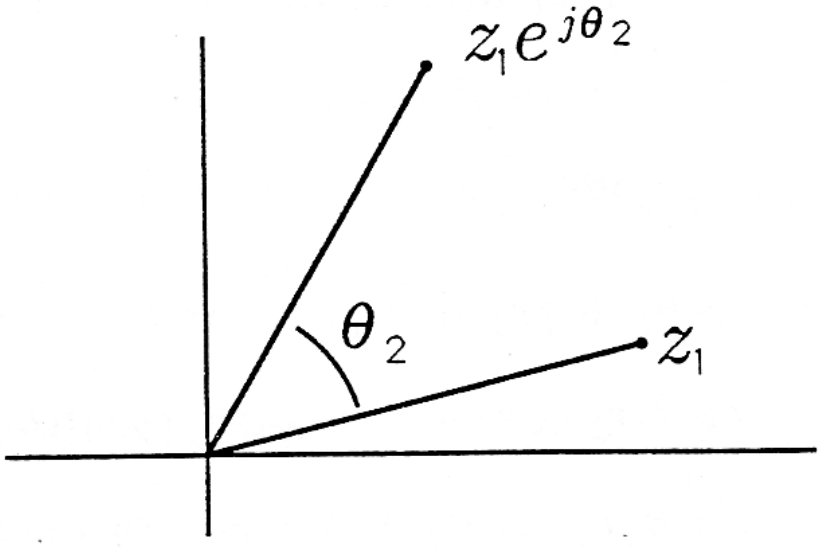
Multiplying Complex Numbers

Rotation. There is a special case of complex multiplication that will become very important in our study of phasors in the chapter on Phasors. When z_1 is the complex number $z_1 = r_1 e^{j\theta_1}$ and z_2 is the complex number $z_2 = e^{j\theta_2}$, then the product of z_1 and z_2 is

Equation:

$$z_1 z_2 = z_1 e^{j\theta_2} = r_1 e^{j(\theta_1+\theta_2)}.$$

As illustrated in [link], $z_1 z_2$ is just a rotation of z_1 through the angle θ_2.



Rotation of Complex Numbers

Exercise:

Problem:

Begin with the complex number $z_1 = x + jy = re^{j\theta}$. Compute the complex number $z_2 = jz_1$ in its Cartesian and polar forms. The complex number z_2 is sometimes called perp(z_1). Explain why by writing perp(z_1) as $z_1 e^{j\theta_2}$. What is θ_2? Repeat this problem for $z_3 = -jz_1$.

Powers. If the complex number z_1 multiplies itself N times, then the result is

Equation:

$$(z_1)^N = r_1^N e^{jN\theta_1}.$$

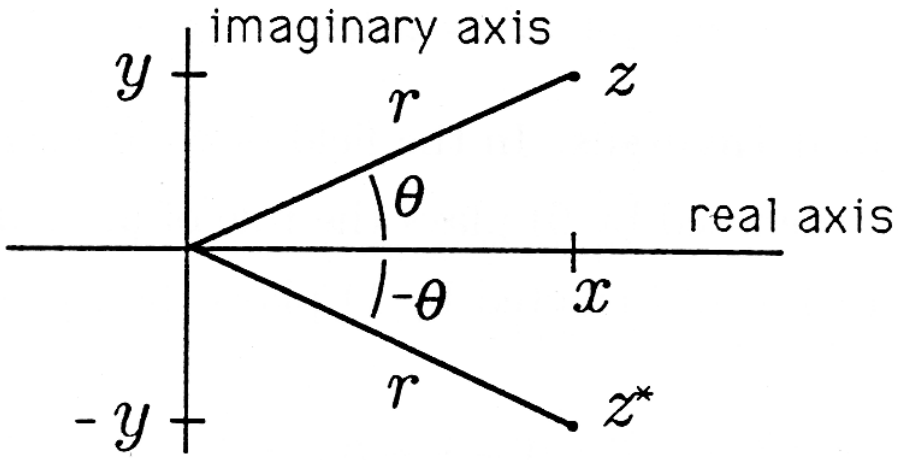
This result may be proved with a simple induction argument. Assume $z_1^k = r_1^k e^{jk\theta_1}$. (The assumption is true for $k = 1$.) Then use the recursion $z_1^{k+1} = z_1^k z_1 = r_1^{k+1} e^{j(k+1)\theta_1}$. Iterate this recursion (or induction) until $k + 1 = N$. Can you see that, as *n* ranges from $n = 1, ..., N$, the angle of z§ranges from θ_1 to $2\theta_1, ...,$ to $N\theta_1$ and the radius ranges from r_1 to $r_1^2, ...,$ to r_1^N ? This result is explored more fully in Problem 1.19.

Complex Conjugate. Corresponding to every complex number $z = x + jy = re^{j\theta}$ is the complex conjugate

Equation:

$$z^* = x - jy = re^{-j\theta}.$$

The complex number z and its complex conjugate are illustrated in [link]. The recipe for finding complex conjugates is to "change $j$to $-j$. This changes the sign of the imaginary part of the complex number.



A Complex Variable and Its Complex Conjugate

Magnitude Squared. The product of *z* and its complex conjugate is called the magnitude squared of *z* and is denoted by $|z|^2$:
Equation:

$$|z|^2 = z^* z = (x - jy)(x + jy) = x^2 + y^2 = re^{-j\theta} re^{j\theta} = r^2.$$

Note that $|z| = r$ is the radius, or magnitude, that we defined in "Geometry of Complex Numbers".
Exercise:

Problem:

Write z^* as $z^* = zw$. Find w in its Cartesian and polar forms.

Exercise:

Problem: Prove that angle $\left(z_2 z_1^*\right) = \theta_2 - \theta_1$.

Exercise:

Problem:

Show that the real and imaginary parts of $z = x + jy$ may be written as

Equation:

$$\mathrm{Re}\,[z] = \frac{1}{2}\quad z + z^*$$

Equation:

$$\mathrm{Im}\,[z] = 2j\quad z - z^*\quad .$$

Commutativity, Associativity, and Distributivity. The complex numbers commute, associate, and distribute under addition and multiplication as follows:

Equation:

$$\begin{aligned} z_1 + z_2 &= z_2 + z_1 \\ z_1 z_2 &= z_2 z_1 \end{aligned}$$

Equation:

$$\begin{aligned} (z_1 + z_2) + z_3 &= z_1 + (z_2 + z_3) \\ z_1(z_2 z_3) &= (z_1 z_2) z_3 \\ z_1(z_2 + z_3) &= z_1 z_2 + z_1 z_3. \end{aligned}$$

Identities and Inverses. In the field of complex numbers, the complex number $0 + j0$ (denoted by 0) plays the role of an additive identity, and the complex number $1 + j0$ (denoted by 1) plays the role of a multiplicative identity:

Equation:

$$\begin{aligned} z+0 &= z = 0+z \\ z1 &= z = 1z. \end{aligned}$$

In this field, the complex number $-z = -x + j(-y)$ is the additive inverse of *z*, and the complex number $z^{-1} = \frac{x}{x^2+y^2} + j \quad \frac{-y}{x^2+y^2}$ is the multiplicative inverse:

Equation:

$$\begin{aligned} z+(-z) &= 0 \\ zz^{-1} &= 1. \end{aligned}$$

Exercise:

Problem:

Show that the additive inverse of $z = re^{j\theta}$ may be written as $re^{j(\theta+\pi)}$.

Exercise:

Problem: Show that the multiplicative inverse of *z* may be written as
Equation:

$$z^{-1} = \frac{1}{z^* z} z^* = \frac{1}{x^2+y^2}\ (x-jy).$$

Show that $z^* z$ is real. Show that z^{-1} may also be written as
Equation:

$$z^{-1} = r^{-1} e^{-j\theta}.$$

Plot *z* and z^{-1} for a representative *z*.

Exercise:

Problem: Prove $(j)^{-1} = -j$.

Exercise:

Problem: Find z^{-1} when $z = 1 + j1$.

Exercise:

Problem:

Prove $\left(z^{-1}\right)^* = \quad z^* \quad ^{-1} = r^{-1}e^{j\theta} = \dfrac{1}{z^* z} z$. Plot z and $\left(z^{-1}\right)^*$ for a representative z.

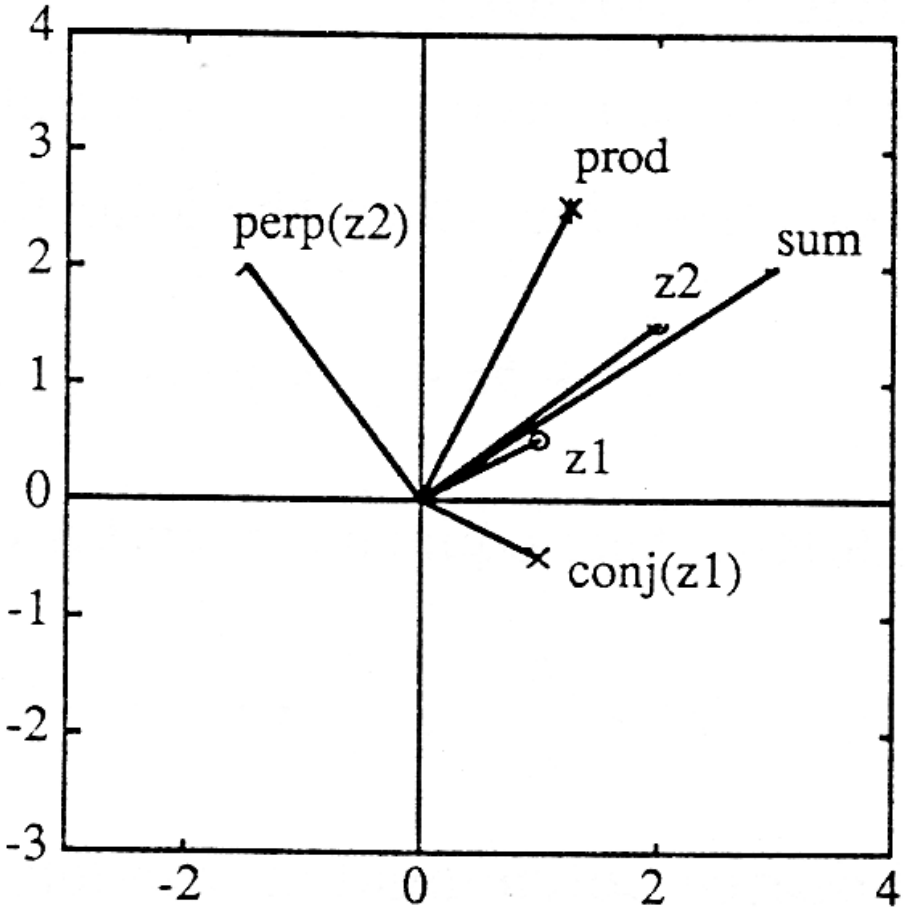
Exercise:

Problem:

Find all of the complex numbers z with the property that $jz = -z^*$. Illustrate these complex numbers on the complex plane.

Demo 1.2 (MATLAB). Create and run the following script file (name it Complex Numbers)[footnote]
If you are using PC-MATLAB, you will need to name your file cmplxnos.m.

```
clear, clg
j=sqrt(-1)
z1=1+j*.5,z2=2+j*1.5
z3=z1+z2,z4=z1*z2
z5=conj(z1),z6=j*z2
avis([-4 4 -4 4]),axis('square'),plot(z1,'0')
hold on
plot(z2,'0'),plot(z3,'+'),plot(z4,'*'),
plot(z2,'0'),plot(z3,'+'),plot(z4,'*'),
plot(z5,'x'),plot(z6,'x')
```



Complex Numbers (Demo 1.2)

With the help of Appendix 1, you should be able to annotate each line of this program. View your graphics display to verify the rules for add, multiply, conjugate, and perp. See [link].

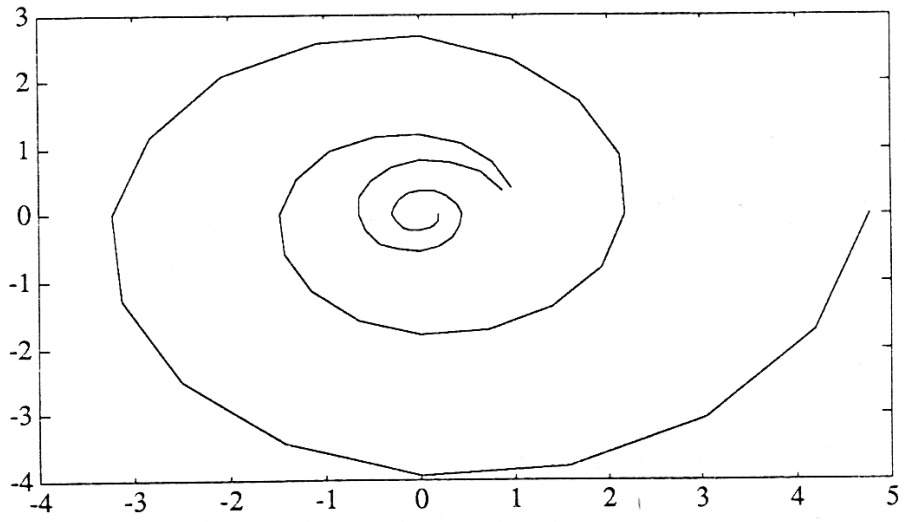
Exercise:

Problem: Prove that $z^0 = 1$.

Exercise:

Problem:

(MATLAB) Choose $z_1 = 1.05e^{j2\pi/16}$ and $z_2 = 0.95e^{j2\pi/16}$. Write a MATLAB program to compute and plot z_1^n and z_2^n for $n = 1, 2, ..., 32$. You should observe a figure like [link].



Powers of z

Complex Numbers: Roots of Quadratic Equations

You probably first encountered complex numbers when you studied values of z (called roots or zeros) for which the following equation is satisfied:
Equation:

$$az^2 + bz + c = 0.$$

For $a \neq 0$ (as we will assume), this equation may be written as
Equation:

$$z^2 + \frac{b}{a}z + \frac{c}{a} = 0.$$

Let's denote the second-degree polynomial on the left-hand side of this equation by $p(z)$:
Equation:

$$p\left(z\right) = z^2 + \frac{b}{a}z + \frac{c}{a}.$$

This is called a *monic polynomial* because the coefficient of the highest-power term $\left(z^2\right)$ is 1. When looking for solutions to the quadratic equation $z^2 + \frac{b}{a}z + \frac{c}{a} = 0$, we are really looking for roots (or zeros) of the polynomial $p(z)$. The fundamental theorem of algebra says that there are two such roots. When we have found them, we may factor the polynomial $p(z)$ as follows:
Equation:

$$p\left(z\right) = z^2 + \frac{b}{a}z + \frac{c}{a} = \left(z - z_1\right)\left(z - z_2\right).$$

In this equation, z_1 and z_2 are the roots we seek. The factored form $p(z) = \left(z - z_1\right)\left(z - z_2\right)$ shows clearly that $p\left(z_1\right) = p\left(z_2\right) = 0$, meaning that the

quadratic equation $p(z) = 0$ is solved for $z = z_1$ and $z = z_2$. In the process of factoring the polynomial $p(z)$, we solve the quadratic equation and vice versa.

By equating the coefficients of z^2, z^1, and z^0 on the left-and right-hand sides of [link], we find that the sum and the product of the roots z_1 and z_2 obey the equations
Equation:

$$\begin{aligned} z_1 + z_2 &= -\tfrac{b}{a} \\ z_1 z_2 &= \tfrac{c}{a} \end{aligned}.$$

You should always check your solutions with these equations.

Completing the Square. In order to solve the quadratic equation $z^2 + \frac{b}{a}z + \frac{c}{a} = 0$ (or, equivalently, to find the roots of the polynomial $z^2 + \frac{b}{a}z + \frac{c}{a}\Big)$, we "complete the square" on the left-hand side of [link]:
Equation:

$$\left(z + \frac{b}{2a}\right)^2 - \left(\frac{b}{2a}\right)^2 + \frac{c}{a} = 0.$$

This equation may be rewritten as
Equation:

$$\left(z + \frac{b}{2a}\right)^2 = \left(\frac{1}{2a}\right)^2 \left(b^2 - 4ac\right).$$

We may take the square root of each side to find the solutions
Equation:

$$z_{1,2} = -\frac{b}{2a} \pm \frac{1}{2a}\sqrt{b^2 - 4ac}.$$

Exercise:

Problem:

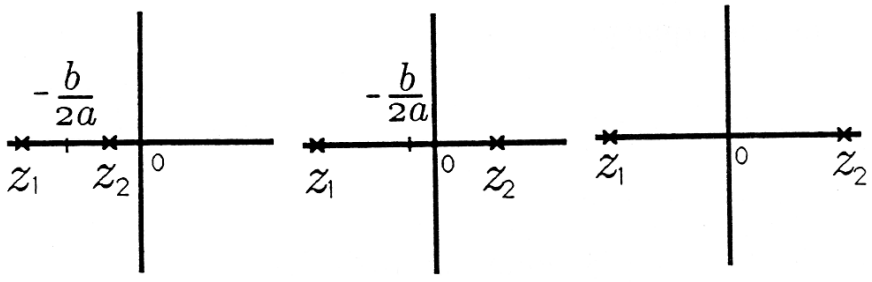
With the roots z_1 and z_2 defined in Equation 1.29, prove that $(z - z_1)(z - z_2)$ is, indeed, equal to the polynomial $z^2 + \frac{b}{a}z + \frac{c}{a}$. Check that $z_1 + z_2 = -\frac{b}{a}$ and $z_1 z_2 = \frac{c}{a}$

In the equation that defines the roots z_1 and z_2, the term $b^2 - 4ac$ is critical because it determines the nature of the solutions for z_1 and z_2. In fact, we may define three classes of solutions depending on $b^2 - 4ac$.

(i) Overdamped $\left(b^2 - 4ac > 0\right)$. In this case, the roots z_1 and z_2 are
Equation:

$$z_{1,2} = -\frac{b}{2a} \pm \frac{1}{2a}\sqrt{b^2 - 4ac}.$$

These two roots are real, and they are located symmetrically about the point $-\frac{b}{2a}$. When $b = 0$, they are located symmetrically about 0 at the points $\pm\frac{1}{2a}\sqrt{-4ac}$. (In this case, $-4ac > 0$.) Typical solutions are illustrated in [link].



Typical Roots in the Overdamped Case; (a)

$b/2a > 0,\ \ 4ac > 0$, (b) $b/2a > 0,\ \ 4ac < 0$, and
(c) $b/2a = 0,\ \ 4ac < 0$

Exercise:

Problem:

Compute and plot the roots of the following quadratic equations:

a. $z^2 + 2z + \frac{1}{2} = 0;$

b. $z^2 + 2a - \frac{1}{2} = 0;$

c. $z^2 - \frac{1}{2} = 0.$

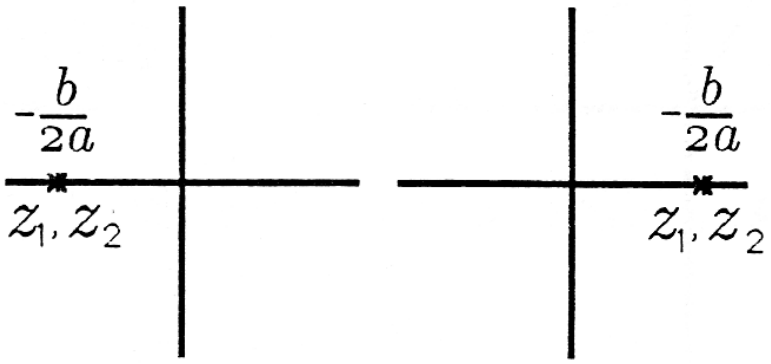
For each equation, check that $z_1 + z_2 = -\frac{b}{a}$ and $z_1 z_2 = \frac{c}{a}$

(ii) Critically Damped $\left(b^2 - 4ac = 0\right)$. In this case, the roots z_1 and z_2 are equal (we say they are repeated):

Equation:

$$z_1 = z_2 = -\frac{b}{2a}.$$

These solutions are illustrated in [link].



Roots in the Critically Damped Case; (a) $b/2a > 0$, and (b) $b/2a < 0$

Exercise:

Problem:

Compute and plot the roots of the following quadratic equations:

a. $z^2 + 2z + 1 = 0$;
b. $z^2 - 2z + 1 = 0$;
c. $z^2 = 0$.

For each equation, check that $z_1 + z_2 = -\dfrac{b}{a}$ and $z_1 z_2 = \dfrac{c}{a}$

(iii) Underdamped $\left(b^2 - 4ac < 0\right)$. The underdamped case is, by far, the most fascinating case. When $b^2 - 4ac < 0$, then the square root in the solutions for z_1 and z_2 ([link]) produces an imaginary number. We may write $b^2 - 4ac$ as $-\left(4ac - b^2\right)$ and write $z_{1,2}$ as

Equation:

$$\begin{aligned} z_{1,2} &= -\tfrac{b}{2a} \pm \tfrac{1}{2a}\sqrt{-(4ac - b^2)} \\ &= -\tfrac{b}{2a} \pm j\tfrac{1}{2a}\sqrt{4ac - b^2}. \end{aligned}$$

These complex roots are illustrated in [link]. Note that the roots are

purely imaginary when $b = 0$, producing the result

Equation:

$$z_{1,2} = \pm j\sqrt{\frac{c}{a}}.$$

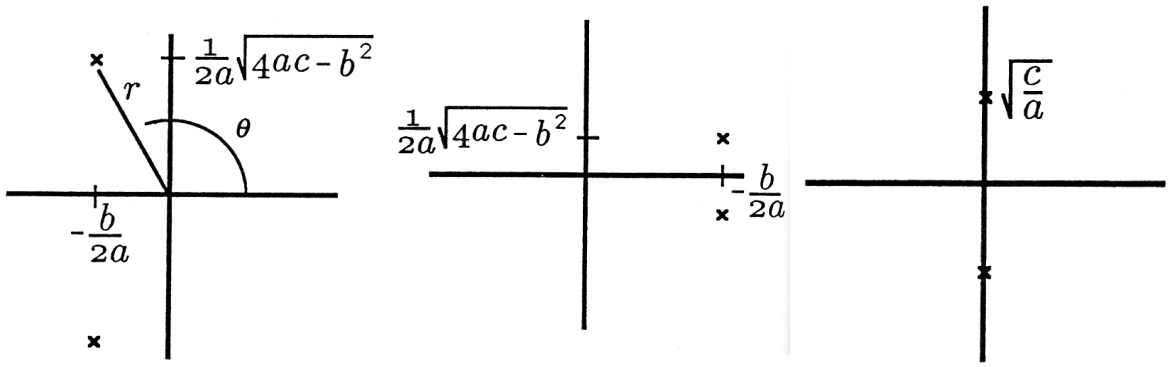


Figure 1.12: Typical Roots in the Underdamped Case; (a) $b/2a > 0$, (b)$b/2a < 0$, and (c) $b/2a = 0$

In this underdamped case, the roots z_1 and z_2 are complex conjugates:
Equation:

$$z_2 = \overset{*}{z_1}.$$

Thus the polynomial $p\left(z\right) = z^2 + \frac{b}{a}z + \frac{c}{a} = \left(z - z_1\right)\left(z - z_2\right)$ also takes the form
Equation:

$$\begin{aligned} p(z) &= \left(z - z_1\right)\left(z - \overset{*}{z_1}\right) \\ &= z^2 - 2\,\mathrm{Re}\left[z_1\right]z + \left|z_1\right|^2. \end{aligned}$$

$\mathrm{Re}[z_1]$ and $\left|z_1\right|^2$ are related to the original coefficients of the polynomial as follows:
Equation:

$$2\,\mathrm{Re}\left[z_1\right] = -\frac{b}{a}$$

Equation:

$$\left|z_1\right|^2 = \frac{c}{a}$$

Always check these equations.

Let's explore these connections further by using the polar representations for z_1 and z_2 :
Equation:

$$z_{1,2} = re^{\pm j\theta}.$$

Then [link] for the polynomial $p(z)$ may be written in the "standard form"
Equation:

$$\begin{aligned} p(z) &= \left(z - re^{j\theta}\right)\left(z - re^{-j\theta}\right) \\ &= z^2 - 2r\ cos\ \theta z + r^2. \end{aligned}$$

[link] is now
Equation:

$$\begin{aligned} 2r\cos\theta &= -\frac{b}{a} \\ r^2 &= \frac{c}{a} \end{aligned}$$

These equations may be used to locate $z_{1,2} = re^{\pm j\theta}$
Equation:

$$\begin{aligned} r &= \sqrt{\frac{c}{a}} \\ \theta &= \pm\cos^{-1}\left(\frac{-b}{\sqrt{4ac}}\right). \end{aligned}$$

Exercise:

Problem:

Prove that $p(z)$ may be written as $p\left(z\right) = z^2 - 2r\cos\theta z + r^2$ in the underdamped case.

Exercise:

Problem:

Prove the relations in [link]. Outline a graphical procedure for locating $z_1 = re^{j\theta}$ and $z_2 = re^{-j\theta}$ from the polynomial $z^2 + \frac{b}{a}z + \frac{c}{a}$.

Exercise:

Problem:

Compute and plot the roots of the following quadratic equations:

a. $z^2 + 2z + 2 = 0$;
b. $z^2 - 2z + 2 = 0$;
c. $z^2 + 2 = 0$.

For each equation, check that $2\,\mathrm{Re}\left[z_{1,2}\right] = -\frac{b}{a}$ and $\left|z_{1,2}\right|^2 = \frac{c}{a}$.

Complex Numbers: Representing Complex Numbers in a Vector Space

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

So far we have coded the complex number $z = x + jy$ with the Cartesian pair (x, y) and with the polar pair $(r\angle\theta)$. We now show how the complex number z may be coded with a two-dimensional vector $\mathbf{z}$ and show how this new code may be used to gain insight about complex numbers.

Coding a Complex Number as a Vector. We code the complex number $z = x + jy$ with the two-dimensional vector $\mathbf{z} = \begin{bmatrix} x \\ y \end{bmatrix}$:

Equation:

$$x + jy = z \Leftrightarrow \mathbf{z} = \begin{bmatrix} x \\ y \end{bmatrix}.$$

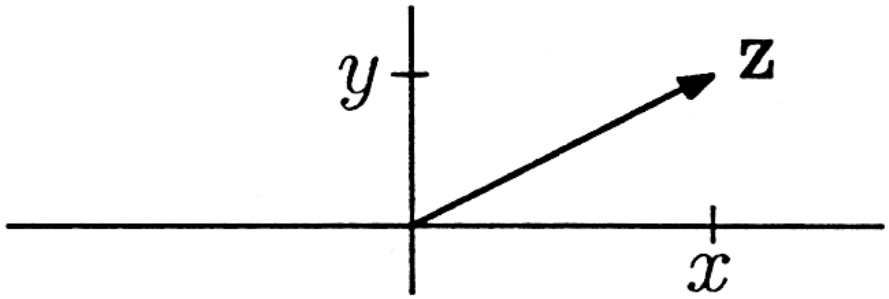
We plot this vector as in [link]. We say that the vector $\mathbf{z}$ belongs to a "vector space." This means that vectors may be added and scaled according to the rules

Equation:

$$\mathbf{z}_1 + \mathbf{z}_2 = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$

Equation:

$$a\mathbf{z} = \begin{bmatrix} ax \\ ay \end{bmatrix}.$$



The Vector z Coding the Complex Number z

Furthermore, it means that an additive inverse $-\mathbf{z}$, an additive identity $\mathbf{0}$, and a multiplicative identity 1 all exist:
Equation:

$$\mathbf{z} + (-\mathbf{z}) = 0$$

Equation:

$$l\mathbf{z} \ = \mathbf{z}.$$

The vector $\mathbf{0}$ is $\mathbf{0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

Prove that vector addition and scalar multiplication satisfy these properties of commutation, association, and distribution:
Equation:

$$\mathbf{z}_1 + \mathbf{z}_2 = z_2 + \mathbf{z}_1$$

Equation:

$$(\mathbf{z}_1 + \mathbf{z}_2) + \mathbf{z}_3 = \mathbf{z}_1 + (\mathbf{z}_2 + \mathbf{z}_3)$$

Equation:

$$a(b\mathbf{z}) = (ab)\mathbf{z}$$

Equation:

$$a\left(\mathbf{z}_1 + \mathbf{z}_2\right) = a\mathbf{z}_1 + a\mathbf{z}_2.$$

Inner Product and Norm. The *inner product* between two vectors $\mathbf{z}_1$ and $\mathbf{z}_2$ is defined to be the real number
Equation:

$$(\mathbf{z}_1, \mathbf{z}_2) = x_1x_2 + y_1y_2.$$

We sometimes write this inner product as the vector product (more on this in Linear Algebra)
Equation:

$$\begin{aligned}(\mathrm{z}_1, \mathrm{z}_2) &= \mathrm{z}_1^T\mathrm{z}_2 \\ &= \begin{bmatrix} x_1 & y_1 \end{bmatrix} \begin{bmatrix} \mathrm{x}_2 \\ \mathrm{y}_2 \end{bmatrix} = \Big(x_1x_2 + y_1y_2 \Big).\end{aligned}$$

Exercise:

Problem: Prove $(\mathbf{z}_1, \mathbf{z}_2) = (\mathbf{z}_2, \mathbf{z}_1)$.

When $\mathbf{z}_1 = \mathbf{z}_2 = \mathbf{z}$, then the inner product between z and itself is the *norm squared* of $\mathbf{z}$:
Equation:

$$||\mathbf{z}||^2 = (\mathbf{z}, \mathbf{z}) = x^2 + y^2.$$

These properties of vectors seem abstract. However, as we now show, they may be used to develop a vector calculus for doing complex arithmetic.

A Vector Calculus for Complex Arithmetic. The addition of two complex numbers z_1 and z_2 corresponds to the addition of the vectors $\mathbf{z}_1$ and $\mathbf{z}_2$:
Equation:

$$z_1 + z_2 \Leftrightarrow \mathbf{z}_1 + \mathbf{z}_2 = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$

The scalar multiplication of the complex number z_2 by the real number x_1 corresponds to scalar multiplication of the vector $\mathbf{z}_2$ by x_1 :
Equation:

$$x_1 z_2 \Leftrightarrow x_1 \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 x_2 \\ x_1 y_2 \end{bmatrix}.$$

Similarly, the multiplication of the complex number z_2 by the real number y_1 is
Equation:

$$y_1 z_2 \leftrightarrow y_1 \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} y_1 x_2 \\ y_1 y_2 \end{bmatrix}.$$

The complex product $z_1 z_2 = (x_1 + j y_1) z_2$ is therefore represented as
Equation:

$$z_1 z_2 \leftrightarrow \begin{bmatrix} x_1 x_2 - y_1 y_2 \\ x_1 y_2 + y_1 x_2 \end{bmatrix}.$$

This representation may be written as the inner product
Equation:

$$z_1 z_2 = z_2 z_1 \leftrightarrow \begin{bmatrix} (\mathbf{v}, \mathbf{z}_1) \\ (\mathbf{w}, \mathbf{z}_1) \end{bmatrix}$$

where $\mathbf{v}$ and $\mathbf{w}$ are the vectors $\mathbf{v} = \begin{bmatrix} x_2 \\ -y_2 \end{bmatrix}$ and $\mathbf{w} = \begin{bmatrix} y_2 \\ x_2 \end{bmatrix}$. By defining the matrix
Equation:

$$\begin{bmatrix} x_2 & -y_2 \\ y_2 & x_2 \end{bmatrix},$$

we can represent the complex product $z_1 z_2$ as a matrix-vector multiply (more on this in Linear Algebra):
Equation:

$$z_1 z_2 = z_2 z_1 \leftrightarrow \begin{bmatrix} x_2 & -y_2 \\ y_2 & x_2 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}.$$

With this representation, we can represent rotation as
Equation:

$$z e^{j\theta} = e^{j\theta} z \leftrightarrow \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

We call the matrix $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ a “rotation matrix.”
Exercise:

Problem: Call $\mathbf{R}(\theta)$ the rotation matrix:
Equation:

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

Show that $\mathbf{R}(-\theta)$ rotates by $(-\theta)$. What can you say about $\mathbf{R}(-\theta)$w when $\mathbf{w} = \mathbf{R}(\theta)\mathbf{z}$?

Exercise:

Problem: Represent the complex conjugate of z as
Equation:

$$z^* \leftrightarrow \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

and find the elements $a, b, c,$ and d of the matrix.

Inner Product and Polar Representation. From the norm of a vector, we derive a formula for the magnitude of z in the polar representation $z = re^{j\theta}$:
Equation:

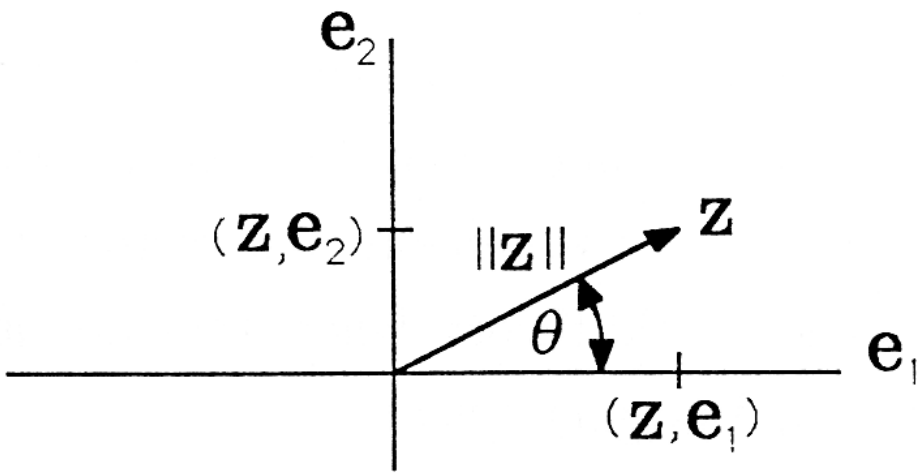
$$r = \left(x^2 + y^2 \right)^{1/2} = ||\mathbf{z}|| = (\mathbf{z}, \mathbf{z})^{1/2}.$$

If we define the coordinate vectors $\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $\mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, then we can represent the vector z as
Equation:

$$\mathbf{z} = (\mathbf{z}, \mathbf{e}_1)\mathbf{e}_1 + (\mathbf{z}, \mathbf{e}_2)\mathbf{e}_2.$$

See [link]. From the figure it is clear that the cosine and sine of the angle θ are
Equation:

$$\cos\theta = \frac{(\mathbf{z}, \mathbf{e}_1)}{||\mathbf{z}||}; \ \sin\theta = \frac{(\mathbf{z}, \mathbf{e}_2)}{||\mathbf{z}||}$$



Representation of z in its Natural Basis

This gives us another representation for any vector z:
Equation:

$$\mathbf{z} = ||\mathbf{z}|| \cos\theta \mathbf{e}_1 + ||\mathbf{z}|| \sin\theta \mathbf{e}_2.$$

The inner product between two vectors z_1 and z_2 is now
Equation:

$$\begin{aligned}(\mathbf{z}_1, \mathbf{z}_2) &= (\mathbf{z}_1, \mathbf{e}_1)\mathbf{e}_1^T (\mathbf{z}_1, \mathbf{e}_2)\mathbf{e}_2^T \begin{bmatrix}(\mathbf{z}_2, \mathbf{e}_1)\mathbf{e}_1 \\ (\mathbf{z}_2, \mathbf{e}_2)\mathbf{e}_2\end{bmatrix} \\ &= (\mathbf{z}_1, \mathbf{e}_1)(\mathbf{z}_2, \mathbf{e}_1) + (\mathbf{z}_1, \mathbf{e}_2)(\mathbf{z}_2, \mathbf{e}_2) \\ &= ||\mathbf{z}_1|| \cos\theta_1 ||\mathbf{z}_2|| \cos\theta_2 + ||\mathbf{z}_1|| \ sin\ \theta_1 ||\mathbf{z}_2|| \sin\theta_2.\end{aligned}$$

It follows that $\cos(\theta_2 - \theta_1) = \cos\theta_2 \cos\theta_1 + \sin\theta_1 \sin\theta_2$ may be written as
Equation:

$$\cos\left(\theta_2 - \theta_1\right) = \frac{\left(\mathbf{z}_1, \mathbf{z}_2\right)}{||\mathbf{z}_1\,||\;\;||\mathbf{z}_2\,||}$$

This formula shows that the cosine of the angle between two vectors $\mathbf{z}_1$ and $\mathbf{z}_2$, which is, of course, the cosine of the angle of $z_2 \overset{*}{z_1}$, is the ratio of the inner product to the norms.

Exercise:

Problem:

Prove the Schwarz and triangle inequalities and interpret them:

Equation:

$$(\text{Schwarz})\;\; \left(\mathbf{z}_1, \mathbf{z}_2\right)^2 \leq ||\mathbf{z}_1||^2\, ||\mathbf{z}_2||^2$$

Equation:

$$(\text{triangle})\;\; I\;\; \mathbf{z}_1 - \mathbf{z}_2\,|| \leq ||\mathbf{z}_1 - \mathbf{z}_3\,|| + ||\mathbf{z}_2 - \;\; \mathbf{z}3\;\; ||.$$

Complex Numbers: An Electric Field Computation

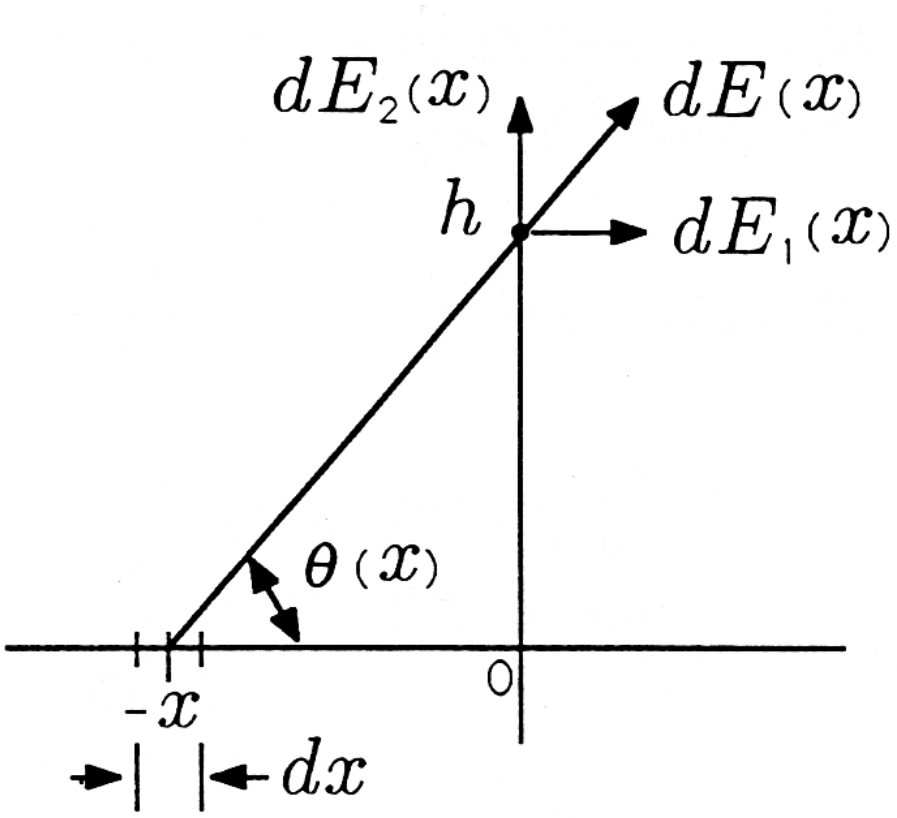
Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

We have established that vectors may be used to code complex numbers. Conversely, complex numbers may be used to code or represent the orthogonal components of any two-dimensional vector. This makes them invaluable in electromagnetic field theory, where they are used to represent the components of electric and magnetic fields.

The basic problem in electromagnetic field theory is to determine the electric or magnetic field that is generated by a static or dynamic distribution of charge. The key idea is to isolate an infinitesimal charge, determine the field set up by this charge, and then to sum the fields contributed by all such infinitesimal charges. This idea is illustrated in [link], where the charge λ, uniformly distributed over a line segment of length dx at point $-x$, produces a field $dE(x)$ at the test point $(0, h)$. The field $dE(x)$ is a "vector" field (as opposed to a "scalar" field), with components $E_1\,(x)$ and $E_2\,(x)$. The intensity or field strength of the field $dE(x)$ is

Equation:

$$|dE\,(x)| = \frac{\lambda dx}{4\pi\epsilon_0\,(h^2 + x^2)}$$



Infinitesimal Charge λdx Producing Field $dE(x)$

But the field strength is directed at angle $\theta(x)$, as illustrated in [link]. The field $dE(x)$ is real with components $dE_1(x)$ and $dE_2(x)$, but we code it as a complex field. We say that the "complex" field at test point $(0, h)$ is
Equation:

$$dE(x) = \frac{\lambda dx}{4\pi\epsilon_0 (h^2 + x^2)} e^{j\theta(x)}$$

with components $dE_1(x)$ and $dE_2(x)$. That is,
Equation:

$$dE(x) = dE_1(x) + j dE_2(x)$$

Equation:

$$dE_1\left(x\right) = \frac{\lambda dx}{4\pi\epsilon_0\left(h^2+x^2\right)}\ cos\ \theta\left(x\right)$$

Equation:

$$dE_2\left(x\right) = \frac{\lambda dx}{4\pi\epsilon_0\left(h^2+x^2\right)}\ sin\ \theta\left(x\right).$$

For charge uniformally distributed with density λ along the x-axis, the total field at the test point $(0, h)$ is obtained by integrating dE:
Equation:

$$\int_{-\infty}^{\infty} dE\left(x\right) = \int_{-\infty}^{\infty} \frac{\lambda}{4\pi\epsilon_0\left(h^2+x^2\right)}\left[\cos\theta\left(x\right) + jsin\theta\left(x\right)\right]dx.$$

The functions cos $\theta(x)$ and sin $\theta(x)$ are
Equation:

$$\cos\theta\left(x\right) = \frac{x}{\left(x^2+h^2\right)^{1/2}};\ \ \sin\theta\left(x\right) = \frac{h}{\left(x^2+h^2\right)^{1/2}}$$

We leave it as a problem to show that the real component E_1 of the field is zero. The imaginary component E_2 is
Equation:

$$E = jE_2 = j\int_{-\infty}^{\infty} \frac{\lambda h}{4\pi\epsilon_0}\frac{dx}{\left(x^2+h^2\right)^{3/2}}$$

Equation:

$$= j\frac{\lambda h}{4\pi\epsilon_0}\frac{x}{h^2\left(x^2+h^2\right)^{1/2}}\Bigg|_{-\infty}^{\infty}$$

Equation:

$$= j\frac{\lambda h}{4\pi\epsilon_0} \quad \frac{1}{h^2} + \frac{1}{h^2} \quad = j\frac{\lambda}{2\pi\epsilon_0 h}$$

Equation:

$$E_2 = \frac{\lambda}{2\pi\epsilon_0 h}.$$

We emphasize that the field at $(0, h)$ is a *real* field. Our imaginary answer simply says that the real field is *oriented* in the vertical direction because we have used the imaginary part of the complex field to code the vertical component of the real field.

Exercise:

> **Problem:**
>
> Show that the horizontal component of the field E is zero. Interpret this finding physically.

From the symmetry of this problem, we conclude that the field around the infinitely long wire of [link] is radially symmetric. So, in polar coordinates, we could say

Equation:

$$E\left(r, \theta\right) = \frac{\lambda}{2\pi\epsilon_0 r}$$

which is independent of θ. If we integrated the field along a radial line perpendicular to the wire, we would measure the voltage difference

Equation:

$$V\left(r_1\right) - V\left(r_0\right) = \int_{r_0}^{r_1} \frac{\lambda}{2\pi\epsilon_0 r} dr = \frac{\lambda}{2\pi\epsilon_0} \quad \log r_1 - \log r_0 \quad .$$

An electric field has units of volts/meter, a charge density λ has units of coulombs/meter, and ϵ_0 has units of coulombs/volt-meter; voltage has units of volts (of course).

Complex Numbers: Numerical Experiment (Quadratic Roots)

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

There is a version of the quadratic equation that will arise over and over again in your study of electrical and mechanical systems:
Equation:

$$s^2 + 2\xi\omega_0 s + \omega_0^2 = 0.$$

For reasons that can only become clear as you continue your study of engineering, the parameter ω_0 is called a *resonant frequency*, and the parameter $\xi \geq 0$ is called a *damping factor*. In this experiment, you will begin by

1. finding the "underdamped" range of values $\xi \geq 0$ for which the roots s_1 and s_2 are complex;
2. finding the "critically damped" value of $\xi \geq 0$ that makes the roots s_1 and s_2 equal; and
3. finding the "overdamped" range of values $\xi \geq 0$ for which s_1 and s_2 are real.
4. For each of these ranges, find the analytical solution for $s_{1,2}$ as a function of ω_0 and ξ; write your solutions in Cartesian and polar forms and present your results as
   Equation:

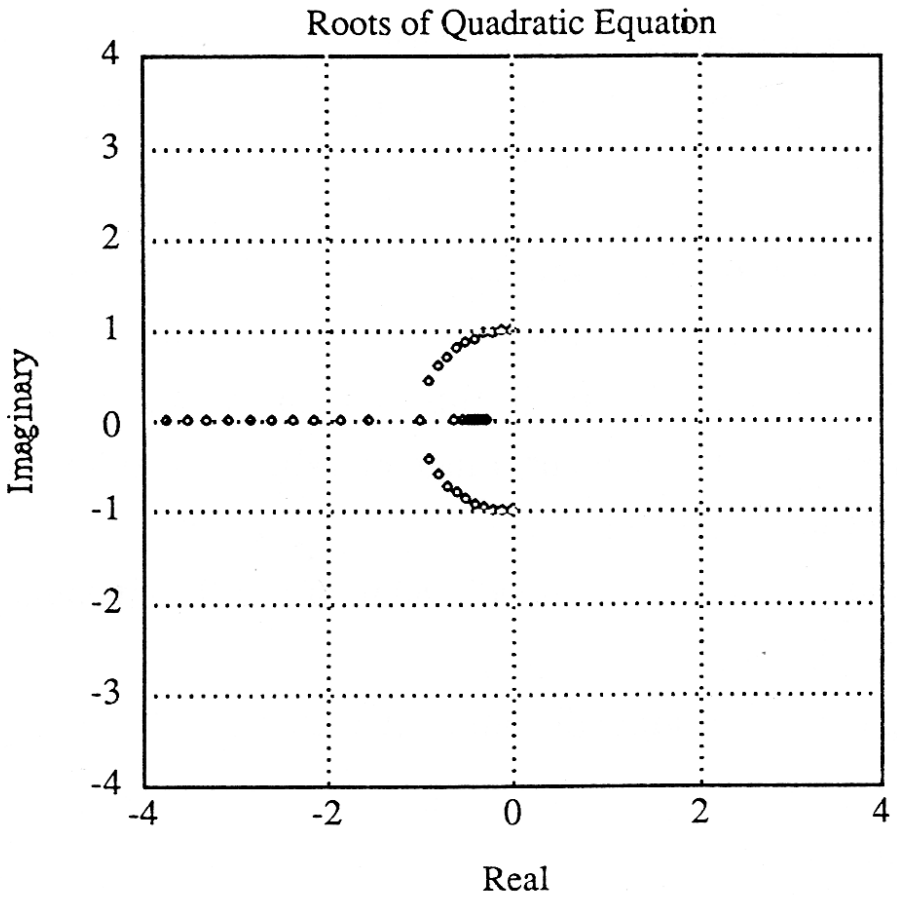
$$s_{1,2} = \begin{matrix} \\ \\ \end{matrix} \begin{matrix} , 0 \leq \xi \leq \xi_c \\ , \xi = \xi_c \\ , \xi \geq \xi_c \end{matrix}$$

where ξ_c is the critically damped value of ξ. Write a MATLAB program that computes and plots $s_{1,2}$ for ω_0 fixed at $\omega_0 = 1$ and ξ variable between 0.0 and 2.0 in steps of 0.1. Interpret all of your findings.

Now organize the coefficients of the polynomial $s^2 + 2\xi s + 1$ into the array $[12\xi1]$. Imbed the MATLAB instructions

```
r=roots([1 2*e 1]);
plot(real(r(1)),imag(r(1)),'o')
plot(real(r(2)),imag(r(2)),'o')
```

in a `for` loop to compute and plot the roots of $s^2 + 2\xi s + 1$ as ξ ranges from 0.0 to 2.0. Note that r is a 1×2 array of complex numbers. You should observe [link]. We call this "half circle and line" the locus of roots for the quadratic equation or the "root locus" in shorthand.



Roots of Quadratic Equation

The Functions e^x and e^jθ: Introduction

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Notes to Teachers and Students

It is essential to write out, term-by-term, every sequence and sum in this chapter. This demystifies the seemingly mysterious notation. The example on compound interest shows the value of limiting arguments in everyday life and gives e^x some real meaning. The function $e^{j\theta}$, covered in the section "The Function of $e^{j\theta}$ and the Unit Circle and "Numerical Experiment (Approximating $e^{j\theta}$, must be understood by all students before proceeding to "Phasors" . The Euler and De Moivre identities provide every tool that students need to derive trigonometric formulas. The properties of roots of unity are invaluable for the study of phasors in "Phasors" .

The MATLAB programs in this chapter are used to illustrate sequences and series and to explore approximations to sin θ and cos θ. The numerical experiment in "Numerical Experiment (Approximating $e^{j\theta}$ illustrates, geometrically and algebraically, how approximations to $e^{j\theta}$ converge.

"Second-Order Differential and Difference Equations" is a little demanding for freshmen, but we give it a once-over-lightly to illustrate the power of quadratic equations and the functions e^x and $e^{j\theta}$. This section also gives a sneak preview of more advanced courses in circuits and systems.

Introduction

It is probably not too strong a statement to say that the function e^x is the most important function in engineering and applied science. In this chapter we study the function e^x and extend its definition to the function $e^{j\theta}$. This study clarifies our definition of $e^{j\theta}$ from "Complex Numbers" and leads us to an investigation of sequences and series. We use the function $e^{j\theta}$ to derive the Euler and De Moivre identities and to produce a number of important trigonometric identities. We define the complex roots of unity and study their partial sums. The results of this chapter will be used in "Phasors" when we study the phasor representation of sinusoidal signals.

The Functions e^x and e^jθ: The Function e^x

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Many of you know the number *e* as the base of the natural logarithm, which has the value 2.718281828459045. . . . What you may not know is that this number is actually defined as the limit of a sequence of approximating numbers. That is,

Equation:

$$e = \lim_{n \to \infty} f_n$$

Equation:

$$f_n = \left(1 + \frac{1}{n}\right)^n, \ n = 1, 2, \ \ldots$$

This means, simply, that the sequence of numbers $(1+1)^1, \left(1+\frac{1}{2}\right)^2, \left(1+\frac{1}{3}\right)^3, \ldots,$ gets arbitrarily close to 2.718281828459045. . . . But why should such a sequence of numbers be so important? In the next several paragraphs we answer this question.

Exercise:

Problem:

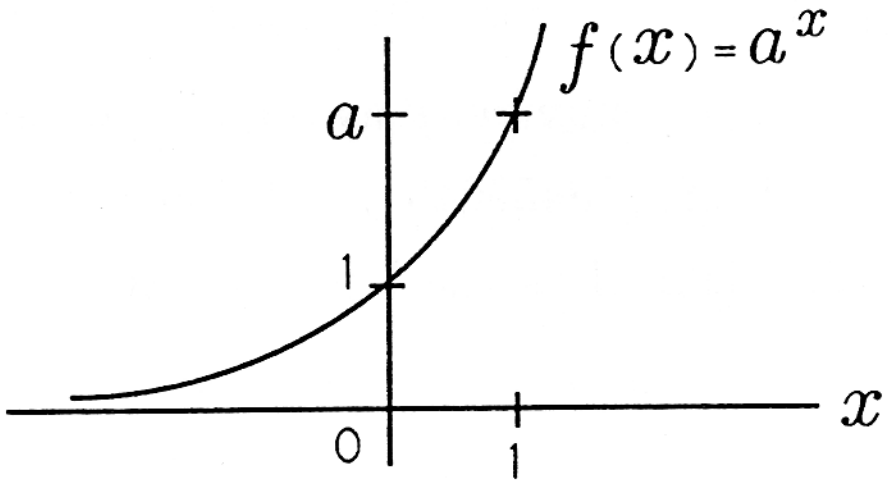
(MATLAB) Write a MATLAB program to evaluate the expression $f_n = \left(1 + \frac{1}{n}\right)^n$ for $n = 1, 2, 4, 8, 16, 32, 64$ to show that $f_n \approx e$ for large n.

Derivatives and the Number e. The number $f_n = \left(1 + \frac{1}{n}\right)^n$ arises in the study of derivatives in the following way. Consider the function
Equation:

$$f(x) = a^x, \ \ a > 1$$

and ask yourself when the derivative of $f(x)$ equals $f(x)$. The function $f(x)$ is plotted in [link] for $a > 1$. The slope of the function at point x is
Equation:

$$\begin{aligned} \frac{df(x)}{dx} &= \lim_{\Delta x \to 0} \frac{a^{x+\Delta x} - a^x}{\Delta x} \\ &= a^x \lim_{\Delta x \to 0} \frac{a^{\Delta x} - 1}{\Delta x}. \end{aligned}$$



The Function $f(x) = a^x$

If there is a special value for *a* such that
Equation:

$$\lim_{\Delta x \to 0} \frac{a^{\Delta x} - 1}{\Delta x} = 1,$$

then $\frac{d}{dx} f(x)$ would equal $f(x)$. We call this value of a the special (or exceptional) number e and write
Equation:

$$f(x) = e^x$$
$$\tfrac{d}{dx} f(x) = e^x.$$

The number e would then be $e = f(1)$. Let's write our condition that $\frac{a^{\Delta x} - 1}{\Delta x}$ converges to 1 as
Equation:

$$e^{\Delta x} - 1 \cong \Delta x, \ \ \Delta x \text{small}$$

or as
Equation:

$$e \cong (1 + \Delta x)^{1/\Delta x}.$$

Our definition of $e = \lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^{1/n}$ amounts to defining $\Delta x = \frac{1}{n}$ and allowing $n \to \infty$ in order to make $\Delta x \to 0$. With this definition for e,

it is clear that the function e^{x} is defined to be $(e)^{x}$:
Equation:

$$e^{x}=\lim_{\Delta x\to 0}\ (1+\Delta x)^{x/\Delta x}.$$

By letting $\Delta x=\dfrac{x}{n}$ we can write this definition in the more familiar form
Equation:

$$e^{x}=\lim_{n\to\infty}\ \left(1+\frac{x}{n}\right)^{n}$$

This is our fundamental definition for the function e^{x}. When evaluated at $x=1$, it produces the definition of e given in [link].

The derivative of e^{x} is, of course,
Equation:

$$\frac{d}{dx}e^{x}=\lim_{n\to\infty}\ n\left(1+\frac{x}{n}\right)^{n-1}\frac{1}{n}=e^{x}.$$

This means that Taylor's theorem[footnote] may be used to find another characterization for e^{x} :
Taylor's theorem says that a function may be completely characterized by all of its derivatives (provided they all exist).
Equation:

$$\begin{aligned} e^{x} &= \textstyle\sum_{n=0}^{\infty}\left[\frac{d^{n}}{dx^{n}}e^{x}\right]_{x=0}\frac{1}{n!}x^{n} \\ &= \textstyle\sum_{n=0}^{\infty}\frac{x^{n}}{n!}. \end{aligned}$$

When this series expansion for e^{x} is evaluated at $x=1$, it produces the following series for e:
Equation:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}.$$

In this formula, $n!$ is the product $n(n-1)(n-2)\cdots(2)1$. Read $n!$ as "n factorial."
Exercise:

> **Problem:** (MATLAB) Write a MATLAB program to evaluate the sum
> **Equation:**
>
> $$S_N = \sum_{n=0}^{N} \frac{1}{n!}$$
>
> for $N = 1, 2, 4, 8, 16, 32, 64$ to show that $S_N \cong e$ for large N. Compare S_{64} with f_{64} from [link]. Which approximation do you prefer?

Compound Interest and the Function $\mathbb{e}^{\mathbb{x}}$. There is an example from your everyday life that shows even more dramatically how the function e^x arises. Suppose you invest V_0 dollars in a savings account that offers $100x\%$ annual interest. (When $x = 0.01$, this is 1%; when $x = 0.10$, this is 10% interest.) If interest is compounded only once per year, you have the simple interest formula for V_1, the value of your savings account after 1 compound (in this case, 1 year):

$V_1 = (1 + x)V_0$. This result is illustrated in the block diagram of Figure 2.2(a). In this diagram, your input fortune V_0 is processed by the "interest block" to produce your output fortune V_1. If interest is compounded monthly, then the annual interest is divided into 12 equal parts and applied 12 times. The compounding formula for V_{12}, the value of your savings after 12 compounds (also 1 year) is
Equation:

$$V_{12} = \left(1 + \frac{x}{12}\right)^{12} V_0.$$

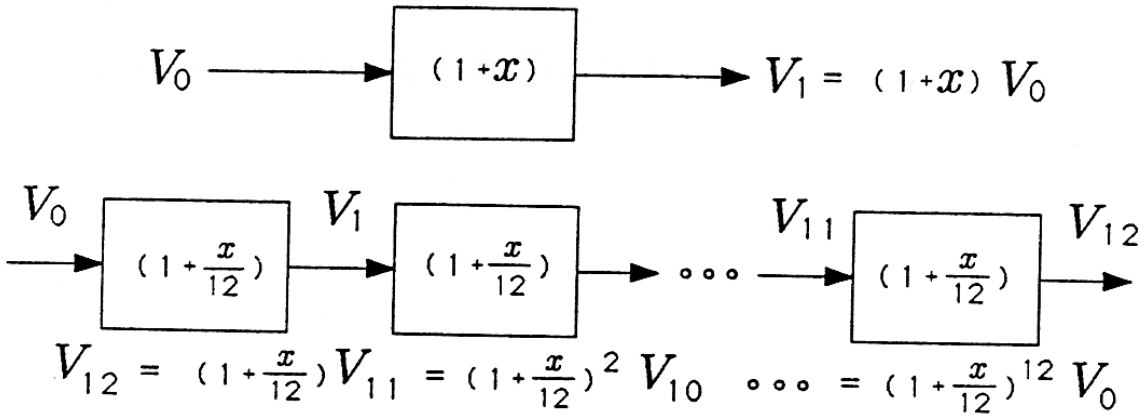
This result is illustrated in [link]. Can you read the block diagram? The general formula for the value of an account that is compounded *n* times per year is

Equation:

$$V_n = \left(1 + \frac{x}{n}\right)^{n} V_0.$$

V_n is the value of your account after n compounds in a year, when the annual interest rate is 100x%.

Block Diagram for Interest Computations; (a) Simple Annual Interest, and (b) Monthly Compounding



Exercise:

Problem:

Verify in [link] that a recursion is at work that terminates at V_n. That is, show that $V_{i+1} = \left(1 + \frac{x}{n}\right) V_1$ for $i = 0, 1, ..., n - 1$ produces the result $V_n = \left(1 + \frac{x}{n}\right)^{n} V_0$.

Bankers have discovered the (apparent) appeal of infinite, or continuous, compounding:
Equation:

$$V_\infty = \lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n V_0.$$

We know that this is just
Equation:

$$V_\infty = e^x V_0.$$

So, when deciding between $100x_1$ % interest compounded daily and $100x_2\%$ interest compounded continuously, we need only compare
Equation:

$$\left(1 + \frac{x_1}{365}\right)^{365} \text{ versus } e^{x_2}.$$

We suggest that daily compounding is about as good as continuous compounding. What do you think? How about monthly compounding?
Exercise:

Problem:

(MATLAB) Write a MATLAB program to compute and plot simple interest, monthly interest, daily interest, and continuous interest versus interest rate $100x$. Use the curves to develop a strategy for saving money.

The Functions e^x and e^jθ: The Function e^jθ and the Unit Circle

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Let's try to extend our definitions of the function e^x to the argument $x = j\Theta$. Then $e^{j\Theta}$ is the function

Equation:

$$e^{j\theta} = \lim_{n\to\infty} \left(1 + j\frac{\theta}{n}\right)^n$$

The complex number $1 + j\frac{\theta}{n}$ is illustrated in [link]. The radius to the point $1 + j\frac{\theta}{n}$ is $r = \left(1 + \frac{\theta^2}{n^2}\right)^{1/2}$ and the angle is $\varphi = \tan^{-1}\frac{\theta}{n}$ This means that the nth power of $1 + j\frac{\theta}{n}$ has radius $r^n = \left(1 + \frac{\theta^2}{n^2}\right)^{n/2}$ and angle $n\varphi = n\tan^{-1}\frac{\theta}{n}$ (Recall our study of powers of z.) Therefore the complex number $\left(1 + j\frac{\theta}{n}\right)^n$ may be written as

Equation:

$$\left(1 + j\frac{\theta}{n}\right)^n = \left(1 + \frac{\theta^2}{n^2}\right)^{n/2}\left[\cos\left(n\tan^{-1}\frac{\theta}{n}\right) + j\sin\left(n\tan^{-1}\frac{\theta}{n}\right)\right].$$

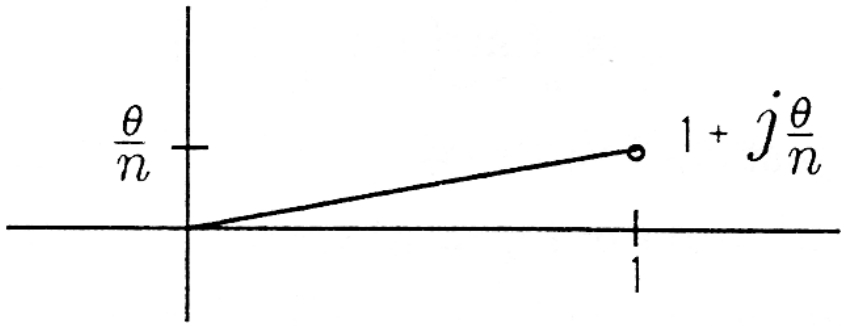
For n large, $\left(1+\frac{\theta^2}{n^2}\right)^{n/2} 2 \cong 1$, and $n \tan^{-1} \frac{\theta}{n} \cong n\frac{\theta}{n} = \theta$. Therefore $\left(1+j\frac{\theta}{n}\right)^n$ is approximately

Equation:

$$\left(1+j\frac{\theta}{n}\right)^n = 1\Big(\cos\theta + j\sin\theta\Big).$$

$\left(1+j\frac{\theta}{n}\right)^n = 1(\cos\theta + j\sin\theta)$.

This finding is consistent with our previous definition of $e^{j\theta}$!



The Complex Number $1 + j\frac{\theta}{n}$

The series expansion for $e^{j\theta}$ is obtained by evaluating Taylor's formula at $x = j\theta$:

Equation:

$$e^{j\theta} = \sum_{n=0}^{\infty} \frac{1}{n!}(j\theta)^n.$$

When this series expansion for $e^{j\theta}$ is written out, we have the formula

Equation:

$$\begin{aligned} e^{j\theta} &= \sum_{n=0}^{\infty} \frac{1}{(2n)!}(j\theta)^{2n} + \sum_{n=0}^{\infty} \frac{1}{(2n+1)!}(j\theta)^{2n+1} \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!}\theta^{2n} + j\sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!}\theta^{2n+1}. \end{aligned}$$

It is now clear that cos θ and sin θ have the series expansions
Equation:

$$\cos\,\theta = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!}\theta^{2n}$$

Equation:

$$\sin\,\theta = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!}\theta^{2n+1}.$$

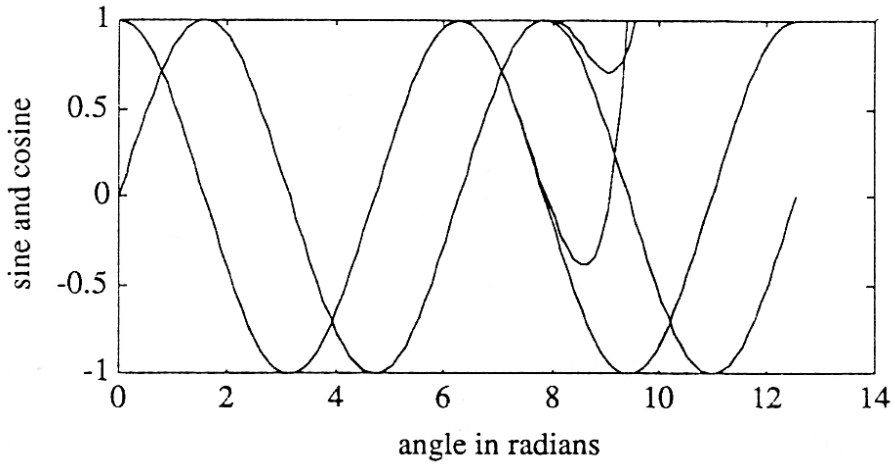
When these infinite sums are truncated at $N-1$, then we say that we have N-term approximations for cos θ and sin θ:
Equation:

$$\cos\,\theta \cong \sum_{n=0}^{N-1} \frac{(-1)^n}{(2n)!}\theta^{2n}$$

Equation:

$$\sin\,\theta \cong \sum_{n=0}^{N-1} \frac{(-1)^n}{(2n+1)!}\theta^{2n+1}.$$

The ten-term approximations to cos θ and sin θ are plotted over exact expressions for cos θ and sin θ in [link]. The approximations are very good over one period $(0 \le \theta \le 2\pi)$, but they diverge outside this interval. For more accurate approximations over a larger range of θ′s, we would need to use more terms. Or, better yet, we could use the fact that cos θ and sin θ are periodic in θ . Then we could subtract as many multiples of 2π as we needed from θ to bring the result into the range $[0, 2\pi]$ and use the ten-term approximations on this new variable. The new variable is called θ-modulo 2π.



Ten-Term Approximations to cos θ and sin θ

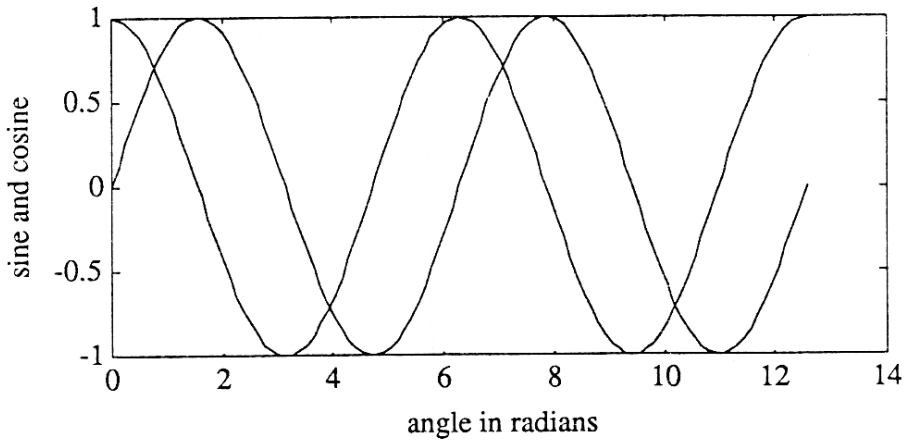
Exercise:

Problem:

Write out the first several terms in the series expansions for cos θ and sin θ.

Demo 2.1 (MATLAB). Create a MATLAB file containing the following demo MATLAB program that computes and plots two cycles of cos θ and sin θ versus θ. You should observe [link]. Note that two cycles take in $2(2\pi)$ radians, which is approximately 12 radians.

```
clg;
j = sqrt(-1);
theta = 0:2*pi/50:4*pi;
s = sin(theta);
c = cos(theta);
plot(theta,s);
elabel('theta in radians');
ylabel('sine and cosine');
hold on
plot(theta,c);
hold off
```



The Functions cos θ and sin θ

Exercise:

Problem:

(MATLAB) Write a MATLAB program to compute and plot the ten-term approximations to cos θ and sin θ for θ running from 0 to $2(2\pi)$ in steps of $2\pi/50$. Compute and overplot exact expressions for cos θ and sin θ. You should observe a result like [link].

The Unit Circle. The *unit circle* is defined to be the set of all complex numbers z whose magnitudes are 1. This means that all the numbers on the unit circle may be written as $z = e^{j\theta}$. We say that the unit circle consists of all numbers generated by the function $z = e^{j\theta}$ as θ varies from 0 to 2π. See [link].

A Fundamental Symmetry. Let's consider the two complex numbers z_1 and $\frac{1}{z_1^*}$, illustrated in [link]. We call $\frac{1}{z_1^*}$ the "reflection of z through the unit circle" (and vice versa). Note that $z_1 = r_1 e^{j\theta_1}$ and $\frac{1}{z_1^*} = \frac{1}{r_1} e^{j\theta_1}$. The complex numbers $z_1 - e^{j\theta}$ and $\frac{1}{z_1^*} - e^{j\theta}$ are illustrated in [link]. The magnitude squared of each is

Equation:

$$\left|z_1 - e^{j\theta}\right|^2 = \left(z_1 - e^{j\theta}\right)\left(\overset{*}{z_1} - e^{-j\theta}\right)$$

Equation:

$$\left|\frac{1}{\overset{*}{z_1}} - e^{j\theta}\right|^2 = \left(\frac{1}{\overset{*}{z_1}} - e^{j\theta}\right)\left(\frac{1}{z_1} - e^{-j\theta}\right).$$

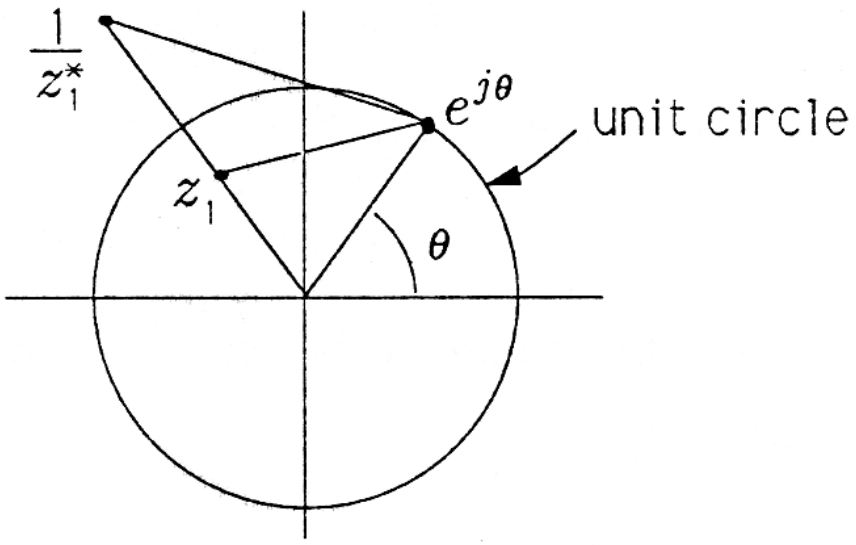
The ratio of these magnitudes squared is
Equation:

$$\beta^2 = \frac{\left(z_1 - e^{j\theta}\right)\left(\overset{*}{z_1} - e^{-j\theta}\right)}{\left(\frac{1}{\overset{*}{z_1}} - e^{j\theta}\right)\left(\frac{1}{z_1} - e^{-j\theta}\right)}$$

This ratio may be manipulated to show that it is independent of θ, meaning that the points z_1 and $\dfrac{1}{\overset{*}{z_1}}$ maintain a constant relative distance from every point on the unit circle:
Equation:

$$\begin{aligned}\beta^2 &= \frac{e^{j\theta}\left(e^{-j\theta}z_1 - 1\right)\left(\overset{*}{z_1}e^{j\theta} - 1\right)e^{-j\theta}}{\frac{1}{zi}\left(1 - e^{j\theta}\overset{*}{z_1}\right)\left(1 - z_1 e^{-j\theta}\right)\frac{1}{z_1}} \\ &= |z_1|^2, \text{independent of} \theta!\end{aligned}$$

This result will be of paramount importance to you when you study digital filtering, antenna design, and communication theory.



The Unit Circle

Exercise:

Problem: Write the complex number $z - e^{j\theta}$ as $re^{j\varphi}$. What are r and φ?

The Functions e^x and e^jθ: The Euler and De Moivre Identities

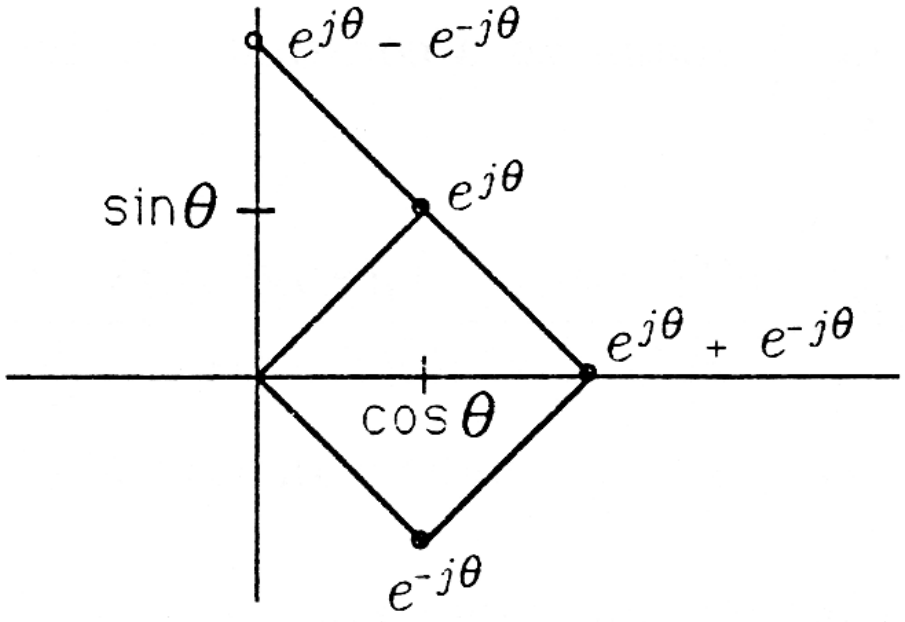
Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

The Euler and De Moivre identities are the fundamental identities for deriving trigonometric formulas. From the identity $e^{j\theta} = \cos\theta + j\sin\theta$ and the conjugate identity $e^{-j\theta} = \left(e^{j\theta}\right)^* = \cos\theta - j\sin\theta$, we have the Euler identities for $\cos\theta$ and $\sin\theta$:

Equation:

$$\begin{aligned} \cos\theta &= \frac{e^{j\theta}+e^{-j\theta}}{2} \\ \sin\theta &= \frac{e^{j\theta}-e^{-j\theta}}{2j} \end{aligned}$$

These identities are illustrated in [link].



Euler's Identities

The identity $e^{j\theta} = \cos\theta + j\sin\theta$ also produces the De Moivre identity:
Equation:

$$\begin{aligned}(\cos\theta + j\ sin\ \theta)^n &= \left(e^{j\theta}\right)^n = e^{jn\theta} \\ &= \cos n\theta + j\sin n\theta.\end{aligned}$$

When the left-hand side of this equation is expanded with the binomial expansion, we obtain the identity
Equation:

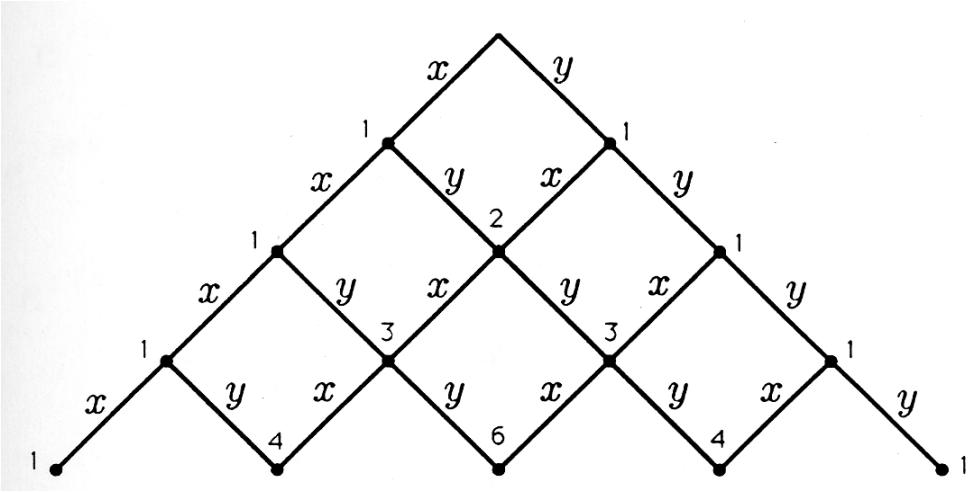
$$\sum_{k=0}^{n}\binom{n}{k}(\cos\theta)^{n-k}(j\sin\theta)^k = \cos n\theta + j\sin n\theta.$$

Binomial Coefficients and Pascal's Triangle. The binomial coefficients $\binom{n}{k}$ in [link] are shorthand for the number

Equation:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}, \quad k = 0, 1, \ \ldots, \ n.$$

This number gives the coefficient of $x^{n-k}y^k$ in the expansion of $(x+y)^n$. How do we know that there are $\binom{n}{k}$ terms of the form $x^{n-k}y^k$? One way to answer this question is to use Pascal's triangle, illustrated in [link]. Each node on Pascal's triangle shows the number of routes that terminate at that node. This number is always the sum of the number of routes that terminate at the nodes just above the node in question. If we think of a left-hand path as an occurrence of an x and a right-hand path as an occurrence of a y, then we see that Pascal's triangle keeps track of the number of occurrences of $x^{n-k}y^k$.



Pascal's Triangle and the Binomial Coefficients

Exercise:

Problem: Prove $\binom{n}{k} = \binom{n}{n-k}$.

Exercise:

Problem: Find an identity for $\binom{n-1}{k-1} + \binom{k}{n-1}$.

Exercise:

Problem: Find "half-angle" formulas for $\cos 2\theta$ and $\sin 2\theta$.

Exercise:

Problem: Show that

A. $\cos 3\theta = \cos^2 \theta - 3\cos \theta \sin^2 \theta$;
B. $\sin 3\theta = 34 \cos^2 \theta \sin \theta - \sin^3 \theta$.

Exercise:

Problem:

Use $e^{j(\theta_1+\theta_2)} = e^{j\theta_1} e^{j\theta_2} = (\cos \theta_1 + j \sin \theta_1)(\cos \theta_2 + j \sin \theta_2)$ to prove

a. $\cos(\theta_1 + \theta_2) = \cos \theta_1 \cos \theta_2 \sin \theta_1 \sin \theta_2$;
b. $\sin(\theta_1 + \theta_2) = \sin \theta_1 \cos \theta_2 + \sin \theta_2 \cos \theta_1$.

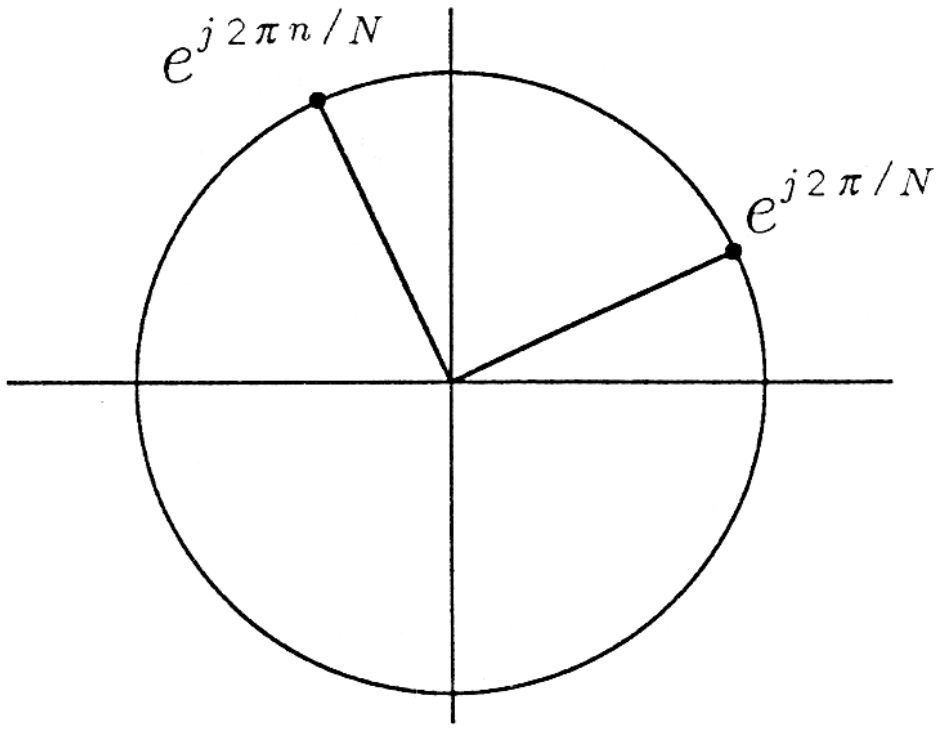
The Functions e^x and e^jθ: Roots of Unity and Related Topics

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

The complex number $z = e^{j2\pi/N}$ is illustrated in [link]. It lies on the unit circle at angle $\theta = 2\pi/N$. When this number is raised to the nth power, the result is $z^n = e^{j2\pi n/N}$. This number is also illustrated in [link]. When one of the complex numbers $e^{j2\pi n/N}$ is raised to the Nth power, the result is
Equation:

$$\left(e^{j2\pi n/N}\right)^N = e^{j2\pi n} = 1.$$

$$(e^{j2\pi n/N})^N = e^{j2\pi n} = 1.$$



The Complex Numbers $e^{j2\pi/N}$ and $e^{j2\pi n/N}$

We say that $e^{j2\pi n/N}$ is one of the Nth roots of unity, meaning that $e^{j2\pi n/N}$ is one of the values of *z* for which

Equation:

$$z^N - 1 = 0.$$

There are *N* such roots, namely,

Equation:

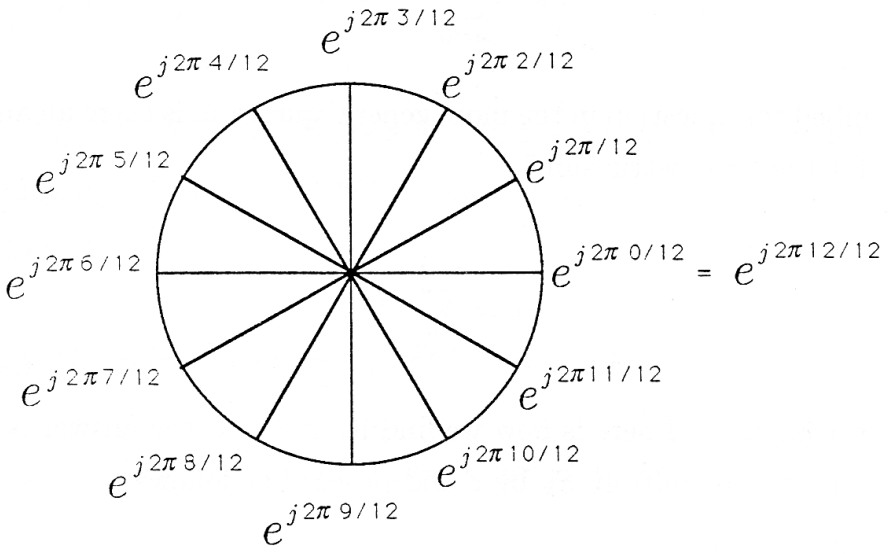
$$e^{j2\pi n/N}, \;\; n = 0, 1, ..., N-1.$$

As illustrated in [link], the 12th roots of unity are uniformly distributed around the unit circle at angles $2\pi n/12$. The sum of all of the Nth roots of unity is zero:

Equation:

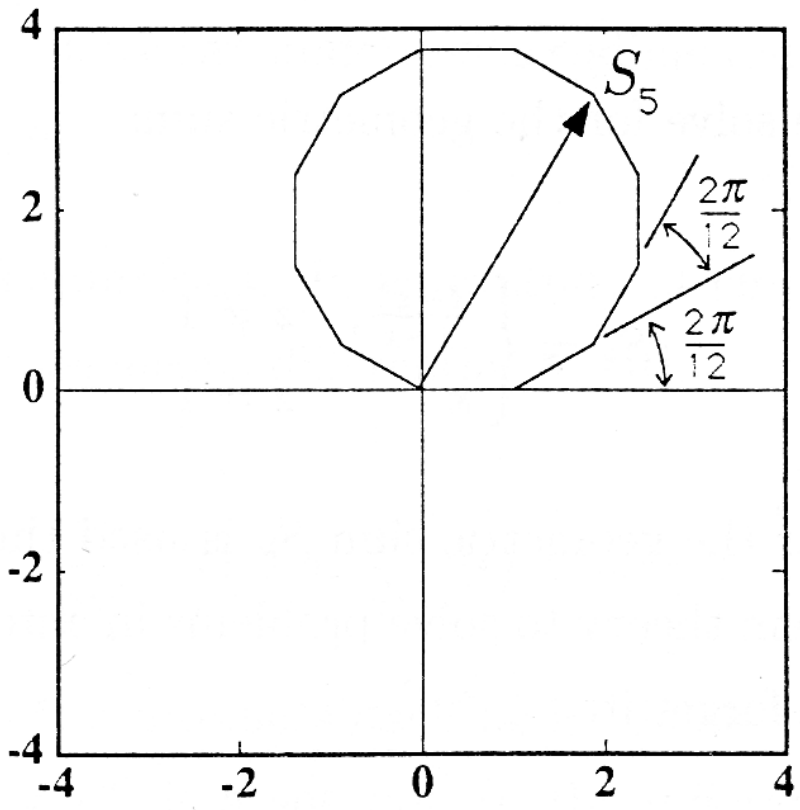
$$S_N = \sum_{n=0}^{N-1} e^{j2\pi n/N} = 0.$$

This property, which is obvious from [link], is illustrated in [link], where the partial sums $S_k = \sum_{n=0}^{k-1} e^{j2\pi n/N}$ are plotted for $k = 1, 2, ..., N$.



Roots of Unity

These partial sums will become important to us in our study of phasors and light diffraction in "Phasors" and in our discussion of filters in "Filtering".



Partial Sums of the Roots of Unity

Geometric Sum Formula. It is natural to ask whether there is an analytical expression for the partial sums of roots of unity:
Equation:

$$S_k = \sum_{n=0}^{k-1} e^{j2\pi n/N}.$$

We can imbed this question in the more general question, is there an analytical solution for the “geometric sum”
Equation:

$$S_k = \sum_{n=0}^{k-1} z^n \ ?$$

The answer is yes, and here is how we find it. If $z = 1$, the answer is $S_k = k$. If $z \neq 1$, we can premultiply S_k by z and proceed as follows:
Equation:

$$\begin{aligned} zS_k &= \textstyle\sum_{n=0}^{k-1} z^{n+1} = \sum_{m=1}^{k} z^m \\ &= \textstyle\sum_{m=0}^{k-1} z^m + z^k - 1 \\ &= S_k + z^k - 1. \end{aligned}$$

From this formula we solve for the geometric sum:
Equation:

$$S_k = \begin{cases} \frac{1-z^k}{1-z} & z \neq 1 \\ k, & z = 1. \end{cases}$$

This basic formula for the geometric sum S_k is used throughout electromagnetic theory and system theory to solve problems in antenna design and spectrum analysis. Never forget it.
Exercise:

Problem: Find formulas for $S_k = \sum_{n=0}^{k-1} e^{jn\theta}$ and for $S_k = \sum_{n=0}^{k-1} e^{j2\pi/Nn}$.

Exercise:

Problem: Prove $\sum_{n=0}^{N-1} e^{j2\pi n/N} = 0.$

Exercise:

Problem:

Find formulas for the magnitude and phase of the partial sum $S_k = \sum_{n=0}^{k-1} e^{j2\pi n/N}$.

Exercise:

Problem:

(MATLAB) Write a MATLAB program to compute and plot the partial sum $S_k = \sum_{n=0}^{k-1} e^{j2\pi n/N}$ for $k = 1, 2, ..., N$. You should observe [link].

Exercise:

Problem: Solve the equation $(z+1)^3 = z^3$.

Exercise:

Problem: Find all roots of the equation $z^3 + z^2 + 3z - 15 = 0$.

Exercise:

Problem:

Find c so that $(1+j)$ is a root of the equation $z^{17} + 2z^{15} - c = 0$.

The Functions e^x and e^jθ: Second-Order Differential and Difference Equations

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

With our understanding of the functions e^x, $e^{j\Theta}$, and the quadratic equation $z^2 + \frac{b}{a}z + \frac{c}{a} = 0$, we can undertake a rudimentary study of differential and difference equations.

Differential Equations. In your study of circuits and systems you will encounter the homogeneous differential equation
Equation:

$$\frac{d^2}{dt^2}x\left(t\right) + a_1\frac{d}{dt}x\left(t\right) + a_2 = 0.$$

Because the function e^{st} reproduces itself under differentiation, it is plausible to assume that $x\left(t\right) = e^{st}$ is a solution to the differential equation. Let's try it:
Equation:

$$\begin{gathered}\frac{d^2}{dt^2}\left(e^{st}\right) + a_1\frac{d}{dt}\left(e^{st}\right) + a_2\left(e^{st}\right) = 0 \\ \left(s^2 + a_1 s + a_2\right)e^{st} = 0.\end{gathered}$$

If this equation is to be satisfied for all t, then the polynomial in s must be zero. Therefore we require
Equation:

$$s^2 + a_1 s + a_2 = 0.$$

As we know from our study of this quadratic equation, the solutions are
Equation:

$$s_{1,2} = -\frac{a_1}{2} \pm \frac{1}{2}\sqrt{a_1^2 - 4a_2}.$$

This means that our assumed solution works, provided $s = s_1$ or s_2. It is a fundamental result from the theory of differential equations that the most general solution for $x(t)$ is a linear combination of these assumed solutions:
Equation:

$$x\left(t\right) = A_1 e^{s_1 t} + A_2 e^{s_2 t}.$$

If $a_1^2 - 4a_2$ is less than zero, then the roots s_1 and s_2 are complex:
Equation:

$$s_{1,2} = -\frac{a_1}{2} \pm j\frac{1}{2}\sqrt{4a_2 - a_1^2}.$$

Let's rewrite this solution as
Equation:

$$s_{1,2} = \sigma \pm j\omega$$

where σ and ω are the constants
Equation:

$$\sigma = -\frac{a_1}{2}$$

Equation:

$$\omega = \frac{1}{2}\sqrt{4a_2 - a_1^2}.$$

With this notation, the solution for $x(t)$ is
Equation:

$$x\left(t\right) = A_1 e^{\sigma t} e^{j\omega t} + A_2 e^{\sigma t} e^{-jt\omega t}.$$

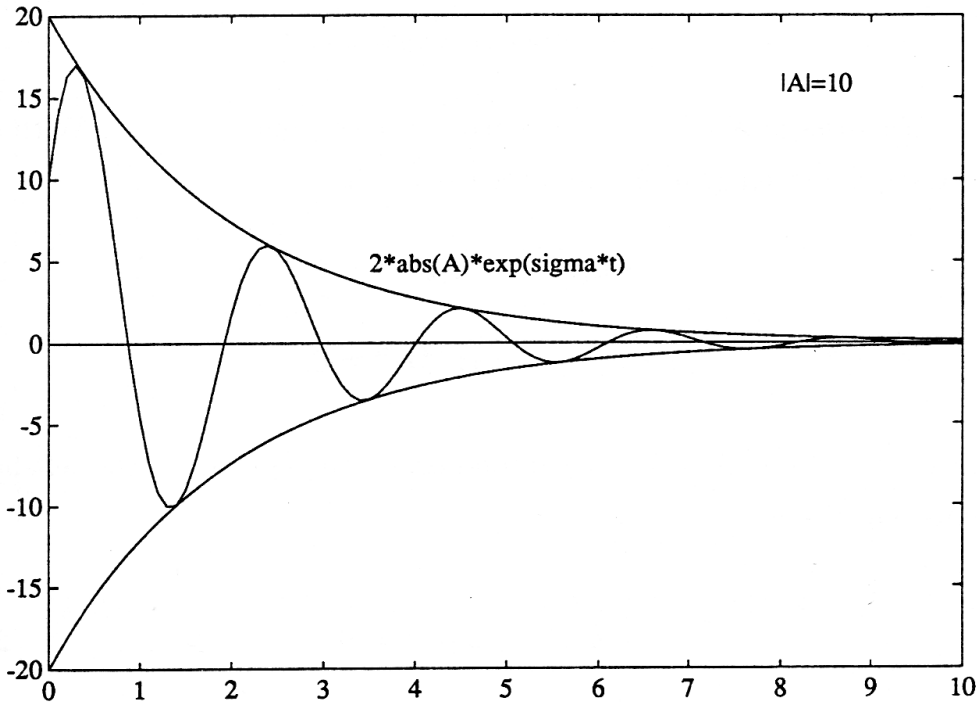
If this solution is to be real, then the two terms on the right-hand side must be complex conjugates. This means that $A_2 = A_1^*$ and the solution for $x(t)$ is
Equation:

$$\begin{aligned} x\left(t\right) &= A_1 e^{\sigma t} e^{j\omega t} + A_1^* e^{\sigma t} e^{-J\omega t} \\ &= 2\,\mathrm{Re}\{A_1 e^{\sigma t} e^{j\omega t}\}. \end{aligned}$$

The constant A_1 may be written as $A_1 = |A| e^{j\varphi}$. Then the solution for $x(t)$ is
Equation:

$$x\left(t\right) = 2|A| e^{\sigma t}\,\cos\left(\omega t + \varphi\right).$$

This “damped cosinusoidal solution” is illustrated in [link].



The Solution to a Second-Order Differential Equation

Exercise:

Problem:

Find the general solutions to the following differential equations:

a. $\frac{d^2}{dt^2}X(t) + 2\frac{d}{dt}x(t) + 2 = 0;$

b. $\frac{d^2}{dt^2}x(t) + 2\frac{d}{dt}x(t) - 2 = 0;$

c. $\frac{d^2}{dt^2}x(t) + 2 = 0.$

Difference Equations. In your study of digital filters you will encounter homogeneous difference equations of the form

Equation:

$$x_n + a_1 x_{n-1} + a_2 x_{n-2} = 0.$$

What this means is that the *sequence* $\{x_n\}$ obeys a *homogeneous recursion*:
Equation:

$$x_n = -a_1 x_{n-1} - a_2 x_{n-2}.$$

A plausible guess at a solution is the geometric sequence $x_n = z^n$. With this guess, the difference equation produces the result
Equation:

$$\begin{aligned} z^n + a_1 z^{n-1} + a_2 z^{n-2} &= 0 \\ \left(1 + a_1 z^{-1} + a_2 z^{-2}\right) z^n &= 0. \end{aligned}$$

If this guess is to work, then the second-order polynomial on the left-hand side must equal zero:

$1 + a_1 z^{-1} + a_2 z^{-2} = 0$
Equation:

$$\begin{aligned} 1 + a_1 z^{-1} + a_2 z^{-2} &= 0 \\ z^2 + a_1 z + a_2 &= 0. \end{aligned}$$

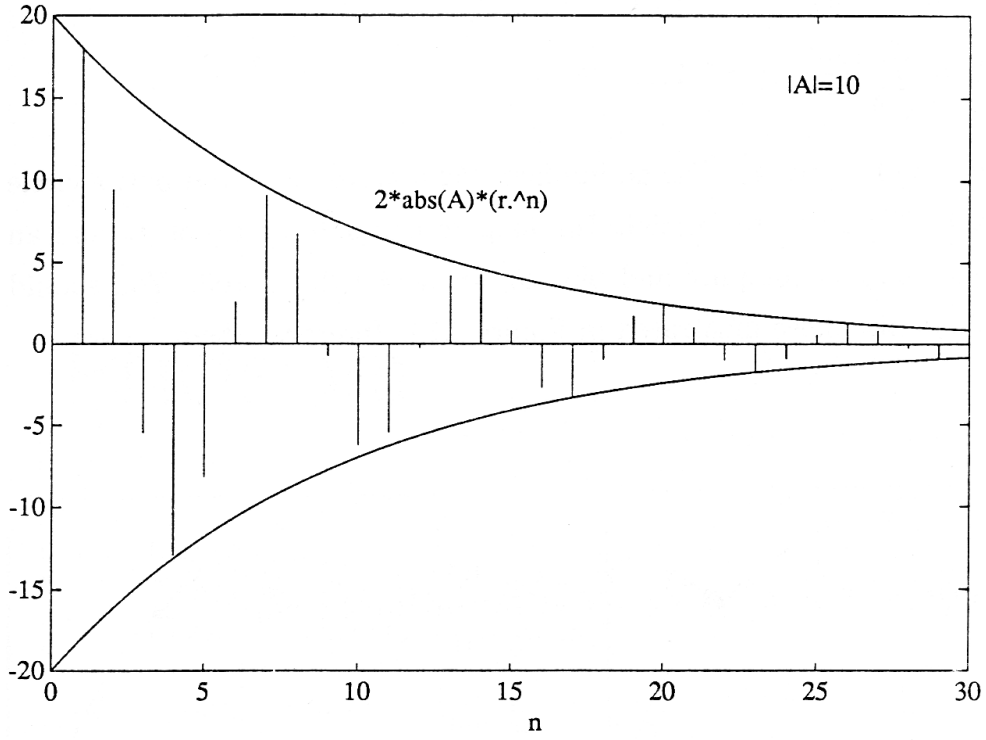
The solutions are
Equation:

$$\begin{aligned} z_{1,2} &= -\tfrac{a_1}{2} \pm j\tfrac{1}{2}\sqrt{4a_2 - a_1^2} \\ &= re^{j\theta}. \end{aligned}$$

The general solution to the difference equation is a linear combination of the assumed solutions:
Equation:

$$
\begin{aligned}
x_n &= A_1 z_1^n + A_2 \left(z_1^*\right)^n \\
&= A_1 z_1^n + A_1^* \left(z_1^*\right)^n \\
&= 2\,\mathrm{Re}\{A_1 z_1^n\} \\
&= 2|A| r^n \cos\left(\theta n + \varphi\right).
\end{aligned}
$$

This general solution is illustrated in [link].



The Solution to a Second-Order Difference Equation

Exercise:

Problem:

Find the general solutions to the following difference equations:

a. $x_n + 2x_{n-1} + 2 = 0$;
b. $x_n - 2x_{n-1} + 2 = 0$;
c. $x_n + 2x_{n-2} = 0$.

The Functions e^x and e^jθ: Numerical Experiment (Approximating e^jθ)

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

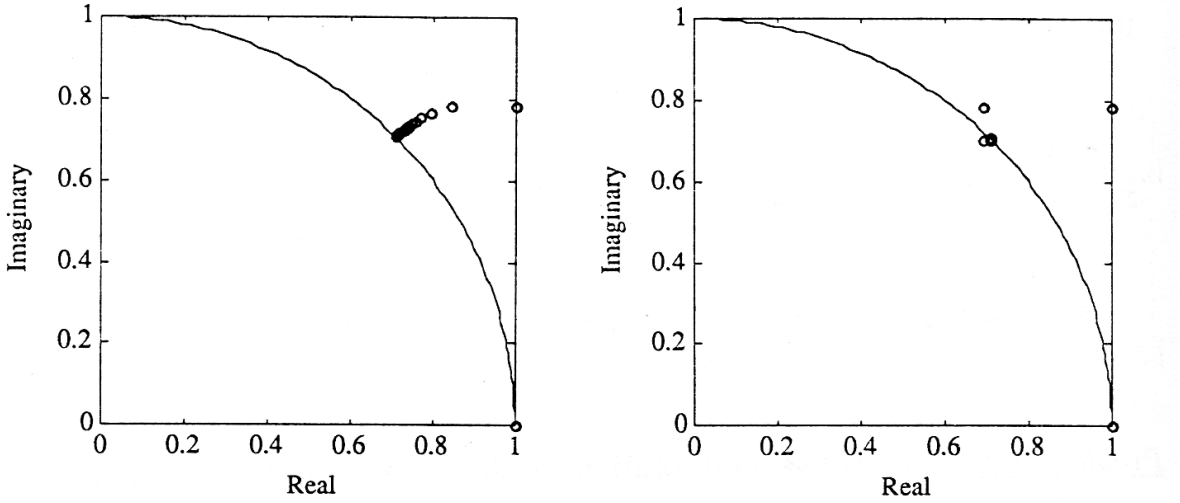
We have demonstrated that the function $e^{j\theta}$ has two representations:

1. $e^{j\theta} = \lim_{n\to\infty} \left(1 + \frac{j\theta}{n}\right)^n$; and
2. $e^{j\theta} = \lim_{n\to\infty} \sum_{k=0}^{n} \frac{(j\theta)^k}{k!}$.

In this experiment, you will write a MATLAB program to evaluate the two functions f_n and S_n for twenty values of n:

1. $f_n = \left(1 + \frac{j\theta}{n}\right)^n, n = 1, 2, ..., 20$; and
2. $S_n = \sum_{k=0}^{n} \frac{(j\theta)^k}{k!}, n = 1, 2, ..., 20k$.

Choose $\theta = \pi/4 (= \text{pi}/4)$. Use an implicit `for` loop to draw and plot a circle of radius 1. Then use an implicit for loop to compute and plot f_n and an explicit for loop to compute and plot S_n for $n = 1, 2, ..., 100$. You should observe plots like those illustrated in [link]. Interpret them.



Plots for Convergence;(a) of f_n, and (b) of S_n

Phasors: Introduction

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Notes to Teachers and Students:

Phasors! For those who understand them, they are of incomparable value for the study of elementary and advanced topics. For those who misunderstand them, they are a constant source of confusion and are of no apparent use. So, let's understand them.

The conceptual leap from the complex number $e^{j\theta}$ to the phasor $e^{j(\omega t+\theta)}$ comes in "Phasor Representation of Signals". Take as long as necessary to understand every geometrical and algebraic nuance. Write the MATLAB program in "Exercise 6" to fix the key ideas once and for all. Then *use* phasors to study beating between tones, multiphase power, and Lissajous figures in "Beating between Tones" through "Lissajous Figures". We usually conduct a classroom demonstration of beating between tones using two phase-locked sources, an oscilloscope, and a speaker. We also demonstrate Lissajous figures with this hardware.

"Sinusoidal Steady State and the Series RLC Circuit" and Light Scattering by a Slit" on sinusoidal steady state and light scattering are too demanding for freshmen but are right on target for sophomores. These sections may be covered in a sophomore course (or a supplement to a sophomore course) or skipped in a freshman course without consequence.

In the numerical experiment in "Numerical Experiment (Interference Patterns)", students compute and plot interference patterns for two sinusoids that are out of phase.

Introduction

In engineering and applied science, three test signals form the basis for our study of electrical and mechanical systems. The *impulse* is an idealized signal that models very short excitations (like current pulses, hammer blows, pile drives, and light flashes). The *step* is an idealized signal that models excitations that are switched on and stay on (like current in a relay that closes or a transistor that switches). The *sinusoid* is an idealized signal that models excitations that oscillate with a regular frequency (like AC power, AM radio, pure musical tones, and harmonic vibrations). All three signals are used in the laboratory to design and analyze electrical and mechanical circuits, control systems, radio antennas, and the like. The sinusoidal signal is particularly important because it may be used to determine the frequency selectivity of a circuit (like a superheterodyne radio receiver) to excitations of different frequencies. For this reason, every manufacturer of electronics test equipment builds sinusoidal oscillators that may be swept through many octaves of Orequency. (Hewlett-Packard was started in 1940 with the famous HP audio oscillator.)

In this chapter we use what we have learned about complex numbers and the function $e^{j\theta}$ to develop a *phasor calculus* for representing and manipulating sinusoids. This calculus operates very much like the calculus we developed in "Complex Numbers" and "The Functions e^{x} and $e^{j\theta}$" for manipulating complex numbers. We ap- ply our calculus to the study of beating phenomena, multiphase power, series *RLC* circuits, and light scattering by a slit.

Phasors: Phasor Representation of Signals

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

There are two key ideas behind the phasor representation of a signal:

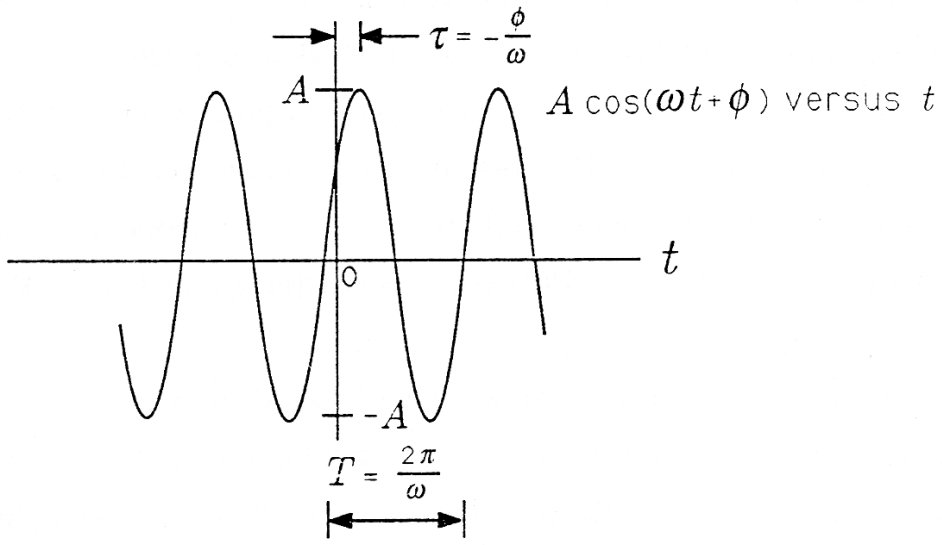
i. a real, time-varying signal may be represented by a complex, time-varying signal; and
ii. a complex, time-varying signal may be represented as the product of a complex number that is *independent* of time and a complex signal that is *dependent* on time.

Let's be concrete. The signal
Equation:

$$x(t) = A\cos\left(\omega t + \varphi\right),$$

illustrated in [link], is a cosinusoidal signal with amplitude A, frequency ω, and phase φ. The amplitude A characterizes the peak-to-peak swing of $2A$, the angular frequency ω characterizes the period $T = \frac{2\pi}{\omega}$ between negative- to-positive zero crossings (or positive peaks or negative peaks), and the phase φ characterizes the time $\tau = \frac{-\varphi}{\omega}$ when the signal reaches its first peak. With τ so defined, the signal $x(t)$ may also be written as
Equation:

$$x(t) = A\cos\omega(t-\tau).$$



A Cosinusoidal Signal

When τ is positive, then τ is a "time delay" that describes the time (greater than zero) when the first peak is achieved. When τ is negative, then τ is a "time advance" that describes the time (less than zero) when the last peak was achieved. With the substitution $\omega = \frac{2\pi}{T}$ we obtain a third way of writing $x(t)$:

Equation:

$$x\left(t\right) = A\cos\frac{2\pi}{T}\left(t-\tau\right).$$

In this form the signal is easy to plot. Simply draw a cosinusoidal wave with amplitude *A* and period *T*; then strike the origin $(t = 0)$ so that the signal reaches its peak at τ. In summary, the parameters that determine a cosinusoidal signal have the following units:

A, arbitrary (e.g., volts or meters/sec, depending upon the application)

ω, in radians/sec (rad/sec)

T, in seconds (sec)

φ, in radians (rad)

τ, in seconds (sec)

Exercise:

Problem:

Show that $x\left(t\right)=A\cos\frac{2\pi}{T}\left(t-\tau\right)$ is “periodic with period T," meaning that $x(t+mT)=x(t)$ for all integer m.

Exercise:

Problem:

The inverse of the period T is called the “temporal frequency” of the cosinusoidal signal and is given the symbol f; the units of $f=\frac{1}{T}$ are $(seconds)^{-1}$ or hertz (Hz). Write $x(t)$ in terms of f. How is f related to ω? Explain why f gives the number of cycles of $x(t)$ per second.

Exercise:

Problem:

Sketch the function $x\left(t\right)=110\cos\left[2\pi\left(60\right)t-\frac{\pi}{8}\right]$ versus t. Repeat for $x\left(t\right)=5\cos\left[2\pi\left(16\times10^{6}\right)t+\frac{\pi}{4}\right]$ and $x\left(t\right)=2\cos\left[\frac{2\pi}{10^{-3}}\left(t-\frac{10^{-3}}{8}\right)\right]$. For each function, determine A,ω,T,f,φ, and τ. Label your sketches carefully.

The signal $x(t)=A\cos\left(\omega t+\varphi\right)$ can be *represented* as the real part of a complex number:

Equation:

$$\begin{aligned} x\left(t\right) &= \mathrm{Re}\left[Ae^{j(\omega t+\varphi)}\right] \\ &= \mathrm{Re}\left[Ae^{j\varphi}e^{j\omega t}\right]. \end{aligned}$$

We call $Ae^{j\varphi}e^{j\omega t}$ the complex representation of $x(t)$ and write
Equation:

$$x\left(t\right) \leftrightarrow Ae^{j\varphi}e^{j\omega t},$$

meaning that the signal $x(t)$ may be reconstructed by taking the real part of $Ae^{j\varphi}e^{j\omega t}$. In this representation, we call $Ae^{j\varphi}$ the *phasor* or complex amplitude representation of $x(t)$ and write
Equation:

$$x\left(t\right) \leftrightarrow Ae^{j\varphi},$$

meaning that the signal $x(t)$ may be reconstructed from $Ae^{j\varphi}$ by multiplying with $e^{j\omega t}$ and taking the real part. In communication theory, we call $Ae^{j\varphi}$ the *baseband* representation of the signal $x(t)$.
Exercise:

Problem:

For each of the signals in Problem 3.3, give the corresponding phasor representation $Ae^{j\varphi}$.

Geometric Interpretation. Let's call
Equation:

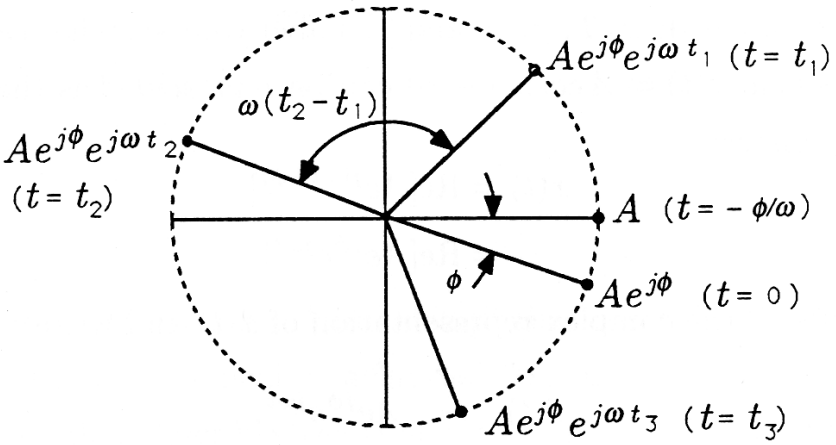
$$Ae^{j\varphi}e^{j\omega t}$$

the complex representation of the real signal $A\cos\left(\omega t+\varphi\right)$. At $t=0$, the complex representation produces the *phasor*
Equation:

$$Ae^{j\varphi}.$$

This phasor is illustrated in [link]. In the figure, φ is approximately $\frac{-\pi}{10}$ If we let t increase to time t_1, then the complex representation produces the phasor

$$Ae^{j\phi}e^{j\omega t_1}.$$



Rotating Phasor

We know from our study of complex numbers that $e^{j\omega t_1}$ just rotates the phasor $Ae^{j\varphi}$ through an angle of ωt_1 ! See [link]. Therefore, as we run t from 0, indefinitely, we rotate the phasor $Ae^{j\varphi}$ indefinitely, turning out the circular trajectory of [link]. When $t = \frac{2\pi}{\omega}$ then $e^{j\omega t} = e^{j2\pi} = 1$.

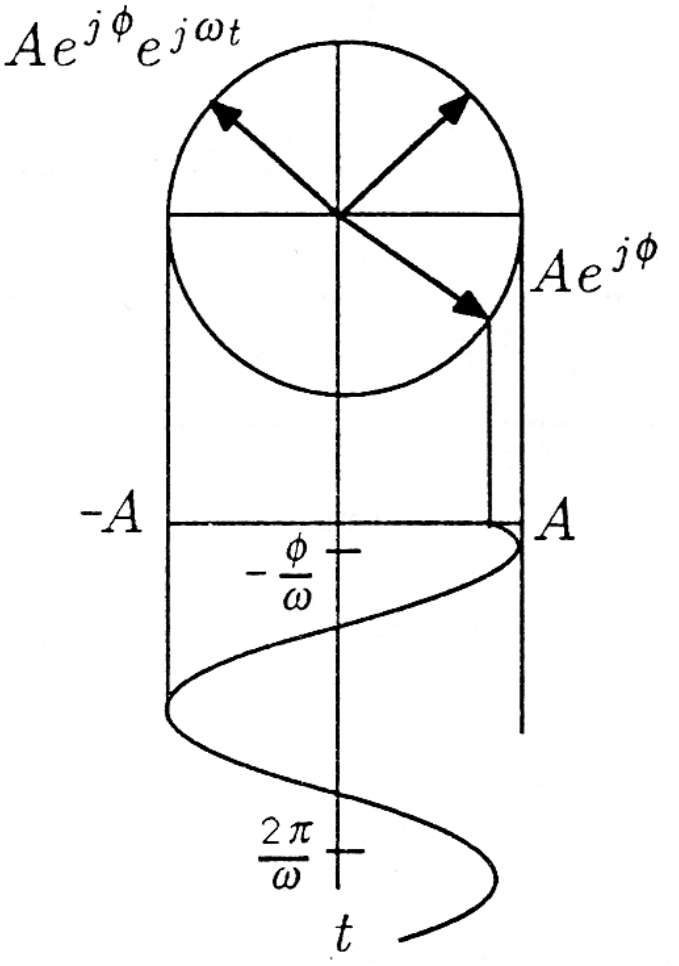
Therefore, every $\left(\frac{2\pi}{\omega}\right)$ seconds, the phasor revisits any given position on the circle of radius *A*. We sometimes call $Ae^{j\varphi}e^{j\omega t}$ a *rotating phasor* whose rotation rate is the frequency ω:

Equation:

$$\frac{d}{dt}\omega t = \omega.$$

This rotation rate is also the frequency of the cosinusoidal signal $A\cos\left(\omega t + \varphi\right)$.

In summary, $Ae^{j\varphi}e^{j\omega t}$ is the complex, or rotating phasor, representation of the signal $A\cos\left(\omega t + \varphi\right)$. In this representation, $e^{j\omega t}$ rotates the phasor $Ae^{j\varphi}$ through angles ωt at the rate ω. The real part of the complex representation is the desired signal $A\cos\left(\omega t + \varphi\right)$. This real part is read off the rotating phasor diagram as illustrated in [link]. In the figure, the angle φ is about $-\frac{2\pi}{10}$. As we become more facile with phasor representations, we will write $x\left(t\right) = \mathrm{Re}\left[Xe^{j\omega t}\right]$ and call $Xe^{j\omega t}$ the complex representation and X the phasor representation. The phasor X is, of course, just the phasor $Ae^{j\varphi}$.



Reading a Real Signal from a Complex, Rotating Phasor

Exercise:

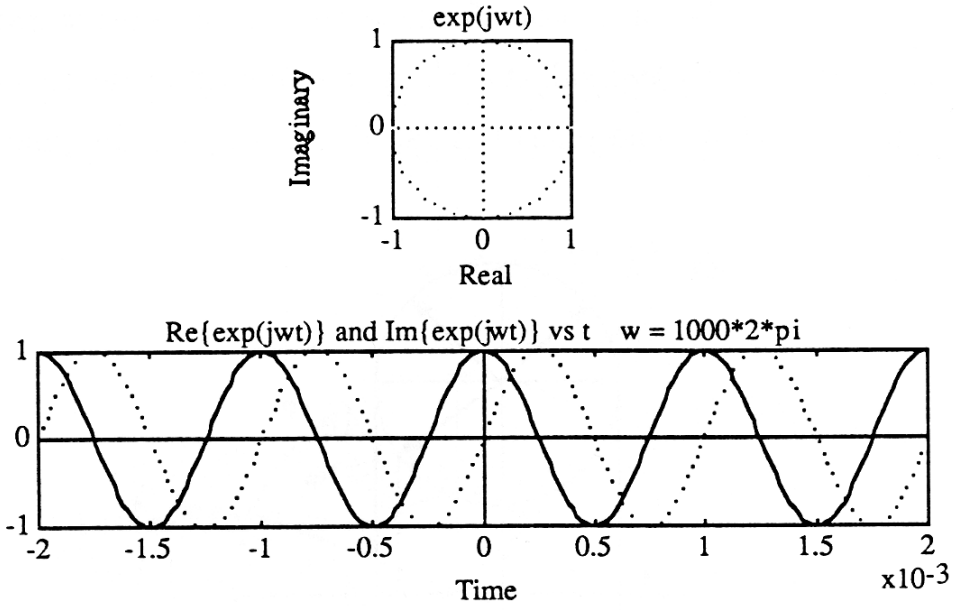
Problem:

Sketch the imaginary part of $Ae^{j\varphi}e^{j\omega t}$ to show that this is $A \sin(\omega t + \varphi)$. What do we mean when we say that the real and imaginary parts of $Ae^{j\varphi}e^{j\omega t}$ are "90° out of phase"?

Exercise:

Problem:

(MATLAB) Modify Demo 2.1 in "The Function e^x and $e^{j\theta}$" so that $\theta = \omega t$, with ω an input frequency variable and t a time variable that ranges from $-2\left(\frac{2\pi}{\omega}\right)$ to $+2\left(\frac{2\pi}{\omega}\right)$ in steps of 0.02 $\left(\frac{2\pi}{\omega}\right)$. In your modified program, compute and plot $e^{j\omega t}$, $\mathrm{Re}\left[e^{j\omega t}\right]$, and $\mathrm{Im}\left[e^{j\omega t}\right]$ for $-2\left(\frac{2\pi}{\omega}\right) \leq t < 2\left(\frac{2\pi}{\omega}\right)$ in steps of 0.02 $\left(\frac{2\pi}{\omega}\right)$. Plot $e^{j\omega t}$ in a two-dimensional plot to get a picture like [link] and plot $\mathrm{Re}\left[e^{j\omega t}\right]$ and $\mathrm{Im}\left[e^{j\omega t}\right]$ versus t to get signals like those of [link]. You should observe something like [link] using the subplot features discussed in An Introduction to MATLAB. (In the figure, w represents Greek ω.)



The Functions $e^{j\omega t}$, $Re\left[e^{j\omega t}\right]$, and $Im\left[e^{j\omega t}\right]$

Positive and Negative Frequencies. There is an alternative phasor representation for the signal $x(t) = A\cos(\omega t + \varphi)$. We obtain it by using

the Euler formula of "The Function e^x s^jθ", namely, cos $\theta = \frac{1}{2}\left(e^{j\theta} + e^{-j\theta}\right)$. When this formula is applied to $x(t)$, we obtain the result

Equation:

$$\begin{aligned} x\left(t\right) &= \frac{A}{2}\left[e^{j(\omega t+\varphi)} + e^{-j(\omega t+\varphi)}\right] \\ &= \frac{A}{2}e^{j\varphi}e^{j\omega t} + \frac{A}{2}e^{-j\varphi}e^{-j\omega t}. \end{aligned}$$

In this formula, the term $\frac{A}{2}e^{j\varphi}e^{j\omega t}$ is a rotating phasor that begins at the phasor value $Ae^{j\varphi}$ (for $t = 0$) and rotates counterclockwise with frequency ω. The term $\frac{A}{2}e^{-j\varphi}e^{-j\omega t}$ is a rotating phasor that begins at the (complex conjugate) phasor value $\frac{A}{2}e^{-j\varphi}$ (for $t = 0$) and rotates clockwise with (negative) ffequency ω. The physically meaningful frequency for a cosine is ω, a positive number like $2\pi(60)$ for 60 Hz power. There is no such thing as a negative frequency. The so-called negative frequency of the term $\frac{A}{2}e^{-j\varphi}e^{-j\omega t}$ just indicates that the direction of rotation for the rotating phasor is clock-wise and not counterclockwise. The notion of a negative frequency is just an artifact of the two-phasor representation of $A \cos\left(\omega t + \varphi\right)$. In the one-phasor representation, when we take the “real part,” the artifact does not arise. In your study of circuits, systems theory, electromagnetics, solid-state devices, signal processing, control, and communications, you will encounter both the one- and two-phasor representations. Become facile with them.

Exercise:

Problem:

Sketch the two-phasor representation of $A \cos\left(\omega t + \varphi\right)$. Show clearly how this representation works by discussing the counterclockwise rotation of the positive frequency part and the clockwise rotation of the negative frequency part.

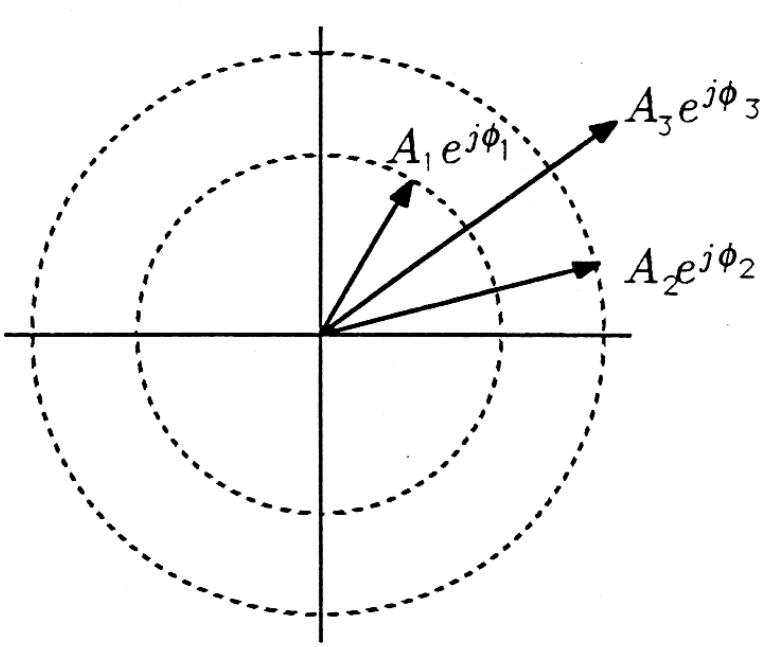
Adding Phasors. The sum of two signals with common frequencies but different amplitudes and phases is
Equation:

$$A_1 \cos(\omega t + \varphi_1) + A_2 \cos(\omega t + \varphi_2).$$

The rotating phasor representation for this sum is
Equation:

$$\left(A_1 e^{j\varphi_1} + A_2 e^{j\varphi_2}\right) e^{j\omega t}.$$

The new phasor is $A_1 e^{j\varphi_1} + A_2 e^{j\varphi_2}$, and the corresponding real signal is $x(t) = \mathrm{Re}\left[\left(A_1 e^{j\varphi_1} + A_2 e^{j\varphi_2}\right) e^{j\omega t}\right]$. The new phasor is illustrated in [link].



Adding Phasors

Exercise:

Problem:

Write the phasor $A_1 e^{j\varphi_1} + A_2 e^{j\varphi_2}$ as $A_3 e^{j\varphi_3}$; determine A_3 and φ_3 in terms of A_1, A_2, φ_1, and φ_2. What is the corresponding real signal?

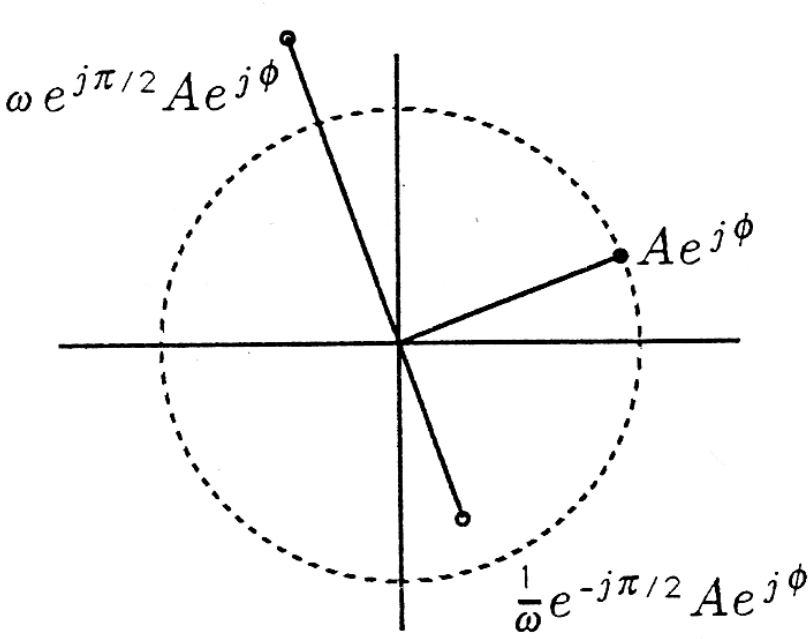
Differentiating and Integrating Phasors. The derivative of the signal $A \cos(\omega t + \varphi)$ is the signal
Equation:

$$\begin{aligned} \tfrac{d}{dt}\ A\ \cos(\omega t + \varphi) &= -\omega A \sin(\omega t + \varphi) \\ &= -\operatorname{Im}\left[\omega A e^{j\varphi} e^{j\omega t}\right] \\ &= \operatorname{Re}\left[j\omega A e^{j\varphi} e^{j\omega t}\right] \\ &= \operatorname{Re}\left[\omega e^{j\pi/2} A e^{j\varphi} e^{j\omega t}\right]. \end{aligned}$$

This finding is very important. It says that the derivative of $A \cos(\omega t + \varphi)$ has the phasor representation
Equation:

$$\begin{aligned} \tfrac{d}{dt} A \cos\left(\omega t + \varphi\right) &\leftrightarrow j\omega A e^{j\varphi} \\ &\leftrightarrow \omega e^{j\pi/2} A e^{j\varphi}. \end{aligned}$$

These two phasor representations are entirely equivalent. The first says that the phasor $Ae^{j\varphi}$ is complex scaled by $j\omega$ to produce the phasor for $\dfrac{d}{dt} A \cos(\omega t + \varphi)$, and the second says that it is scaled by ω and phased by $+\pi/2$. The phasor representations of $A \cos(\omega t + \varphi)$ and $\dfrac{d}{dt} A \cos(\omega t + \varphi)$ are illustrated in Figure 3.6. Note that the derivative "leads by $\pi/2$ radians (90)."



Differentiating and Integrating Phasors

The integral of $A \cos(\omega t + \varphi)$ is
Equation:

$$\begin{aligned}
\int A \cos(\omega t + \varphi) dt &= \frac{A}{\omega} \sin(\omega t + \varphi) \\
&= \operatorname{Im}\left[\frac{A}{\omega} e^{j\varphi} e^{j\omega t}\right] \\
&= \operatorname{Re}\left[-j\frac{A}{\omega} e^{j\varphi} e^{j\omega t}\right] \\
&= \operatorname{Re}\left[\frac{A}{j\omega} e^{j\varphi} e^{j\omega t}\right] \\
&= \operatorname{Re}\left[\frac{1}{\omega} e^{-j\pi/2} A e^{j\varphi} e^{j\omega t}\right].
\end{aligned}$$

This finding shows that the integral of $A \cos(\omega t + \varphi)$ has the phasor representation
Equation:

$$\begin{aligned} \int A\cos\left(\omega t+\varphi\right)dt \quad &\leftrightarrow \quad \tfrac{1}{j\omega}Ae^{j\varphi} \\ &\leftrightarrow \quad \tfrac{1}{\omega}e^{-j\pi/2}Ae^{j\varphi}. \end{aligned}$$

The phasor $Ae^{j\varphi}$ is complex scaled by $\dfrac{1}{j\omega}$ or scaled by $\dfrac{1}{\omega}$ and phased by $e^{-j\pi/2}$ to produce the phasor for $\displaystyle\int A\cos\left(\omega t+\varphi\right)dt$. This is illustrated in [link]. Note that the integral "lags by $\pi/2$ radians (90). Keep these geometrical pictures of leading and lagging by $\pi/2$ in your mind at all times as you continue your more advanced study of engineering.

An Aside: The Harmonic Oscillator. The signal $A\cos\left(\omega t+\varphi\right)$ stands on its own as an interesting signal. But the fact that it reproduces itself (with scaling and phasing) under differentiation means that it obeys the second-order differential equation of the simple harmonic oscillator.[footnote] That is, the differential equation
This means, also, that we have an easy way to synthesize cosines with circuits that obey the equation of a simple harmonic oscillator!
Equation:

$$\frac{d^2x\left(t\right)}{dt^2}+\omega^2x\left(t\right)=0$$

has the solution
Equation:

$$x(t)=A\cos\left(\omega t+\varphi\right).$$

Try it:
Equation:

$$\frac{d^2}{dt^2}x\left(t\right)=\frac{d}{dt}\left[-A\omega\sin\left(\omega t+\varphi\right)\right]=-\omega^2A\cos\left(\omega t+\varphi\right).$$

The constants A and φ are determined from the initial conditions
Equation:

$$x\left(0\right)=A\cos\varphi \qquad\qquad x^{2}\left(0\right)+x^{2}\left(\tfrac{\pi}{2\omega}\right)=A^{2}$$

$$\Leftrightarrow$$

$$x\left(\tfrac{\pi}{2\omega}\right)=-A\sin\varphi \qquad\qquad -\tfrac{x(\pi/2\omega)}{x(0)}=\tan\varphi.$$

Exercise:

Problem:

Show how to compute A and φ in the equation
$x(t)=A\cos\left(\omega t+\varphi\right)$ from the initial conditions $x(\mathrm{O})$ and
$\frac{d}{dt}x\left(t\right)|_{t=0}$.

Phasors: Beating between Tones

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

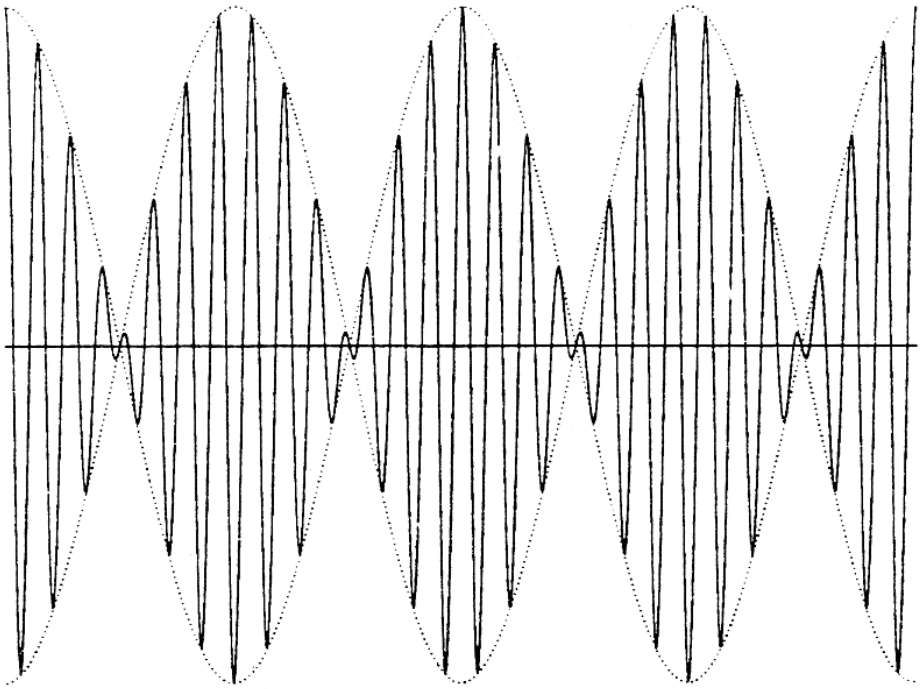
Perhaps you have heard two slightly mistuned musical instruments play pure tones whose frequencies are close but not equal. If so, you have sensed a beating phenomenon wherein a pure tone seems to wax and wane. This waxing and waning tone is, in fact, a tone whose frequency is the average of the two mismatched frequencies, amplitude modulated by a tone whose "beat" frequency is half the difference between the two mismatched frequencies. The effect is illustrated in [link]. Let's see if we can derive a mathematical model for the beating of tones.

We begin with two pure tones whose frequencies are $\omega_0 + \nu$ and $\omega_0 - \nu$ (for example, $\omega_0 = 2\pi \times 10^3 \text{rad}/\text{ sec}$ and $\nu = 2\pi \text{ rad } / \text{ sec}$). The average frequency is ω_0, and the difference frequency is 2ν. What you hear is the sum of the two tones:
Equation:

$$x(t) = A_1 \cos[(\omega_0 + \nu)t + \varphi_1] + A_2 \cos[(\omega_0 - \nu)t + \varphi_2].$$

The first tone has amplitude A_1 and phase φ_1 ; the second has amplitude A_2 and phase φ_2. We will assume that the two amplitudes are equal to A. Furthermore, whatever the phases, we may write them as
Equation:

$$\begin{aligned} \varphi_1 = \varphi + \psi \quad &\text{and} \quad \varphi_2 = \varphi - \psi \\ \varphi = \tfrac{1}{2}(\varphi_1 + \varphi_2) \quad &\text{and} \quad \psi = \tfrac{1}{2}(\varphi_1 - \varphi_2). \end{aligned}$$



Beating between Tones

Recall our trick for representing $x(t)$ as a complex phasor:
Equation:

$$\begin{aligned} x\,(t) &= A\ \mathrm{Re}\ \ e^{j[(\omega_0+\nu)t+\varphi+\psi]}, +, e^{j[(\omega_0-\nu)t+\varphi-\psi]} \\ &= A\ \mathrm{Re}\ \ e^{j(\omega_0 t+\varphi)},\ \ e^{j(\nu t+\psi)} + e^{-j(\nu t+\psi)} \\ &= 2A\ \mathrm{Re}\ \ e^{j(\omega_0 t+\varphi)}, \cos, (\nu t+\psi) \\ &= 2A\ \cos\,(\omega_0 t+\varphi)\cos\,(\nu t+\psi). \end{aligned}$$

This is an amplitude modulated wave, wherein a low frequency signal with beat frequency ν rad/sec modulates a high frequency signal with carrier frequency ω_0 rad/sec. Over short periods of time, the modulating term $\cos\,(\nu t+\psi)$ remains essentially constant while the carrier term $\cos\,(\omega_0 t+\varphi)$ turns out many cycles of its tone. For example, if t runs

from 0 to $\frac{2\pi}{10\nu}$ (about 0.1 seconds in our example), then the modulating wave turns out just 1/10 cycle while the carrier turns out $10\nu\omega_{\Delta}$ cycles (about 100 in our example). Every time νt changes by 2π radians, then the modulating term goes from a maximum (a wax) through a minimum (a wane) and back to a maximum. This cycle takes

Equation:

$$\nu t = 2\pi \Leftrightarrow t = \frac{2\pi}{\nu}\text{seconds},$$

which is 1 second in our example. In this 1 second the carrier turns out 1000 cycles.

Exercise:

Problem:

Find out the frequency of A above middle C on a piano. Assume two pianos are mistuned by $\pm 1\text{Hz}(\pm 2\pi\text{rad}/\text{sec})$. Find their beat frequency ν and their carrier frequency ω_0.

Exercise:

Problem: (MATLAB) Write a MATLAB program to compute and plot

$A\ \cos\ [(\omega_0 + \nu)t + \varphi_1]$, $A\ \cos\ [(\omega_0 - \nu)t + \varphi_2]$, and their sum. Then compute and plot $2A\ \cos\ (\omega_0 t + \varphi)\ \cos\ (\nu t + \psi)$.

Verify that the sum equals this latter signal.

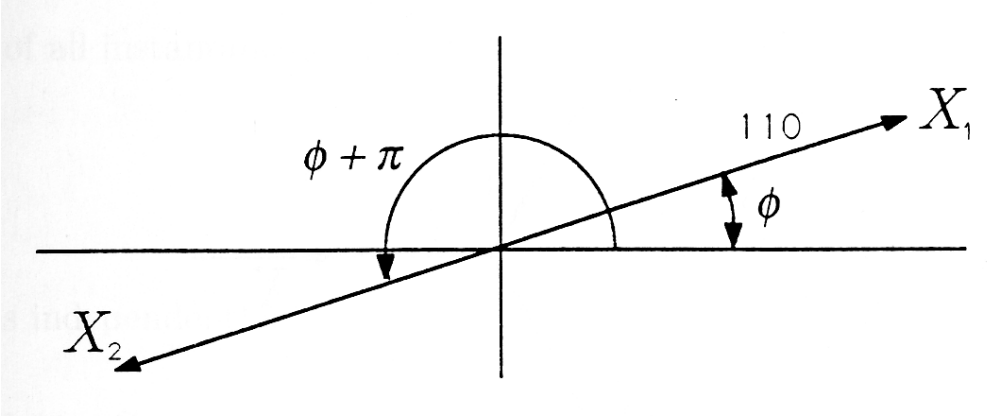
Phasors: Multiphase Power

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

The electrical service to your home is a two-phase service.[footnote] This means that two 110 volt, 60 Hz lines, plus neutral, terminate in the panel. The lines are π radians (180) out of phase, so we can write them as
It really is, although it is said to be “single phase” because of the way it is picked off a single phase of a primary source. You will hear more about this in circuits and power courses.
Equation:

$$\begin{aligned} x_1(t) &= 110\ cos\ [2\pi(60)t+\varphi] = \mathrm{Re}\left\{110e^{j[2\pi(60)t+\varphi]}\right\} \\ &= \mathrm{Re}\left\{X_1 e^{j2\pi(60)t}\right\} \\ &\qquad X_1 = 110e^{j\varphi} \\ x_2(t) &= 110\ cos\ [2\pi(60)t+\varphi+\pi] = \mathrm{Re}\left\{110e^{j[2\pi(60)t+\varphi+\pi]}\right\} \\ &= \mathrm{Re}\left\{X_2 e^{j2\pi(60)t}\right\} \\ &\qquad X_2 = 110e^{j(\varphi+\pi)}. \end{aligned}$$

These two voltages are illustrated as the phasors X_1 and X_2 in [link].



Phasors in Two-Phase Power

You may use $x_1(t)$ to drive your clock radio or your toaster and the difference between $x_1(t)$ and $x_2(t)$ to drive your range or dryer:
Equation:

$$x_1(t) - x_2(t) = 220 \cos[2\pi(60)t + \varphi].$$

The phasor representation of this difference is
Equation:

$$X_1 - X_2 = 220e^{j\varphi}.$$

The breakers in a breaker box span the x_1-to-neutral bus for 110 volts and the x_1-to-x_2 buses for 220 volts.
Exercise:

> **Problem:** Sketch the phasor $X_1 - X_2$ on [link].
>
> Most industrial installations use a three-phase service consisting of the signals $x_1(t), x_2(t)$, and $x_3(t)$:
> **Equation:**

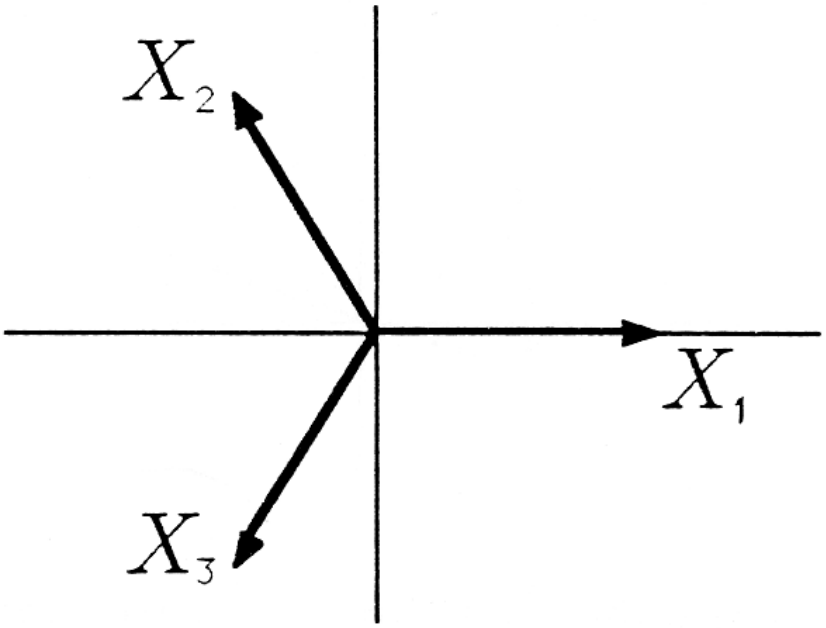
$$x_n(t) = 110\,\mathrm{Re}\left\{e^{j[\omega_0 t + n(2\pi/3)]}\right\} \leftrightarrow X_n = 110e^{jn(2\pi/3)},\ n = 1, 2, 3.$$

The phasors for three-phase power are illustrated in [link].

Exercise:

Problem:

Sketch the phasor $X_2 - X_1$ corresponding to $x_2(t) - x_1(t)$ on [link]. Compute the voltage you can get with $x_2(t) - x_1(t)$. This answer explains why you do not get 220 volts in three-phase circuits. What do you get?



Three-Phase Power

Constant Power. Two- and three-phase power generalizes in an obvious way to N-phase power. In such a scheme, the N signals $x_n(n = 0, 1, ..., N-1)$ are

Equation:

$$\begin{aligned} x_n(t) &= A\cos\left(\omega t + \frac{2\pi}{N}n\right) \\ &= \mathrm{Re}\left[Ae^{j2\pi n/N}e^{j\omega t}\right] \;\leftrightarrow\; X_n = Ae^{j2\pi n/N}. \end{aligned}$$

The phasors X_n are $Ae^{j2\pi(n/N)}$. The sum of all N signals is zero:
Equation:

$$\begin{aligned} \sum_{n=0}^{N-1} x_n(t) &= \mathrm{Re}\left\{A\sum_{n=0}^{N-1} e^{j2\pi n/N}e^{j\omega t}\right\} \\ &= \mathrm{Re}\left\{A\frac{1-e^{j2\pi}}{1-e^{j2\pi/N}}e^{j\omega t}\right\} \\ &= 0. \end{aligned}$$

But what about the sum of the instantaneous powers? Define the instantaneous power of the nth signal to be
Equation:

$$\begin{aligned} p_n(t) = x_n^2(t) &= A^2\cos^2\left(\omega t + \tfrac{2\pi}{N}n\right) \\ &= \tfrac{A^2}{2} + \tfrac{A^2}{2}\cos\left(2\omega t + 2\tfrac{2\pi}{N}n\right) \\ &= \tfrac{A^2}{2} + \mathrm{Re}\left\{\tfrac{A^2}{2}e^{j(2\pi/N)2n}e^{j2\omega t}\right\}. \end{aligned}$$

The sum of all instantaneous powers is (see [link])
Equation:

$$P = \sum_{n=0}^{N-1} p_n(t) = N\frac{A^2}{2},$$

and this is independent of time!
Exercise:

Problem:

Carry out the computations of Equation 3.33 to prove that instantaneous power P is constant in the N-phase power scheme.

Phasors: Lissajous Figures

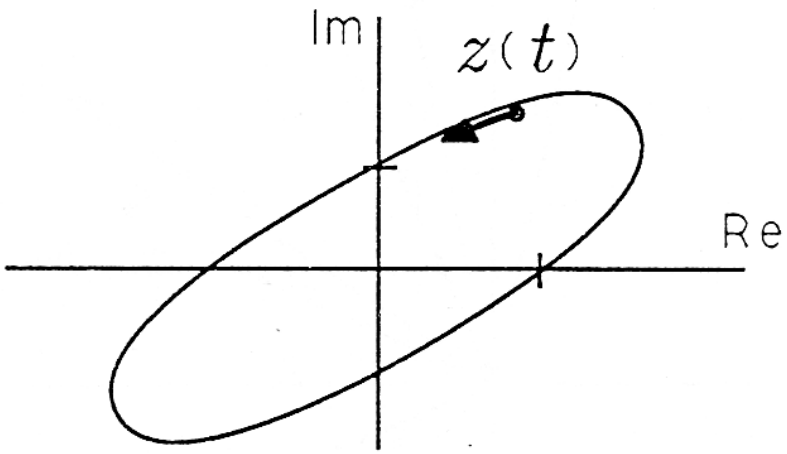
Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Lissajous figures are figures that are turned out on the face of an oscilloscope when sinusoidal signals with different amplitudes and different phases are applied to the time base (real axis) and deflection plate (imaginary axis) of the scope. The electron beam that strikes the phosphorous face then had position

Equation:

$$z(t) = A_x \cos(\omega t + \varphi_x) + jA_y \cos(\omega t + \varphi_y).$$

In this representation, $A_x \cos(\omega t + \varphi_x)$ is the "x-coordinate of the point," and $A_y \cos(\omega t + \varphi)$ is the "y-coordinate of the point." As time runs from 0 to infinity, the point $z(t)$ turns out a trajectory like that of [link]. The figure keeps overwriting itself because $z(t)$ repeats itself every $\frac{2\pi}{\omega}$ seconds. Do you see why?



Lissajous Figure on Oscilloscope Screen

Exercise:

Problem:

Find the intercepts that the Lissajous figure makes with the real and imaginary axes in [link]. At what values of time are these intercepts made?

Exercise:

Problem:

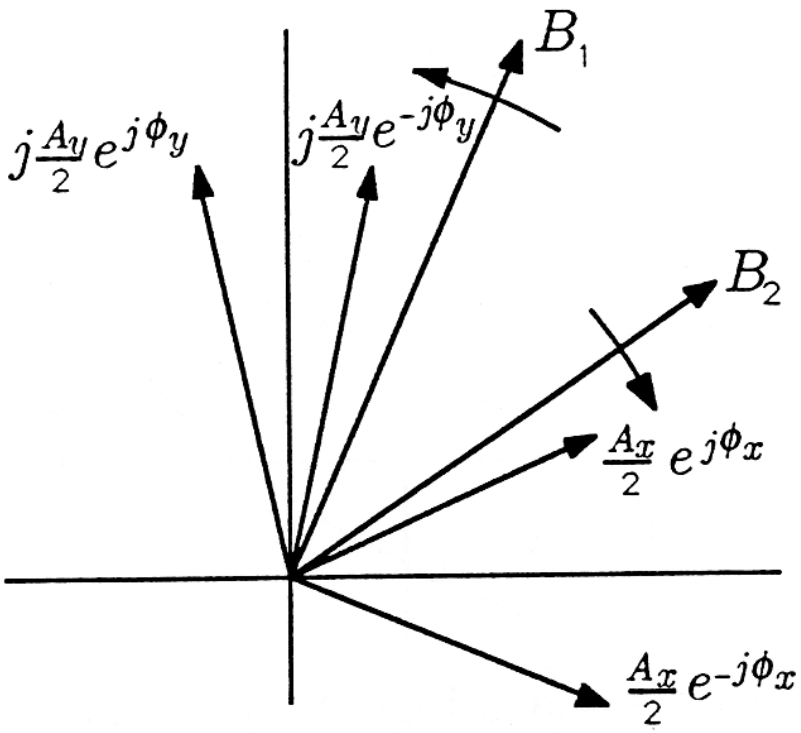
Show that the Lissajous figure $z\left(t\right)=A_x\cos\left(\omega t+\varphi_x\right)+jA_y\cos\left(\omega t+\varphi_y\right)$ is just the rotating phasor $Ae^{j(\omega t+\varphi)}$ when $A_x=A_y=A$, $\varphi_x=\varphi$, and $\varphi_y=\varphi+\frac{\pi}{2}$.

Two-Phasor Representation. We gain insight into the shape of the Lissajous figure if we use Euler's formulas to write $z(t)$ as follows:
Equation:

$$
\begin{aligned}
z(t) &= \frac{A_x}{2}\ e^{j(\omega t+\varphi_x)} + e^{-j(\omega t+\varphi_x)}\ + j\frac{A_y}{2}\ e^{j(\omega t+\varphi_y)} + e^{-j(\omega t+\varphi_y)} \\
&= \frac{A_x e^{j\varphi_x} + jA_y e^{j\varphi_y}}{2}\ e^{j\omega t} + \frac{A_x e^{-j\varphi_x} + jA_y e^{-j\varphi_y}}{2}\ e^{-j\omega t}.
\end{aligned}
$$

This representation is illustrated in [link]. It consists of two *rotating* phasors, with respective phasors B_1 and B_2 :
Equation:

$$
\begin{aligned}
z(t) &= B_1 e^{j\omega t} + B_2 e^{-j\omega t} \\
B_1 &= \frac{A_x e^{j\varphi_x} + jA_y e^{j\varphi_y}}{2} \\
B_2 &= \frac{A_x e^{-j\varphi_x} + jA_y e^{-j\varphi_y}}{2}
\end{aligned}
$$



Two-Phasor Representation of a Lissajous Figure

As t increases, the phasors rotate past each other where they constructively add to produce large excursions of $z(t)$ from the origin, and then they rotate to antipodal positions where they destructively add to produce near approaches of $z(t)$ to the origin.

In electromagnetics and optics, the representations of $z(t)$ given in [link] and [link] are called, respectively, *linear* and *circular* representations of elliptical polarization. In the linear representation, the x- and y-components of z vary along the horizontal and vertical lines. In the circular representation, two phasors rotate in opposite directions to turn out circular trajectories whose sum produces the same effect.

Exercise:

Problem:

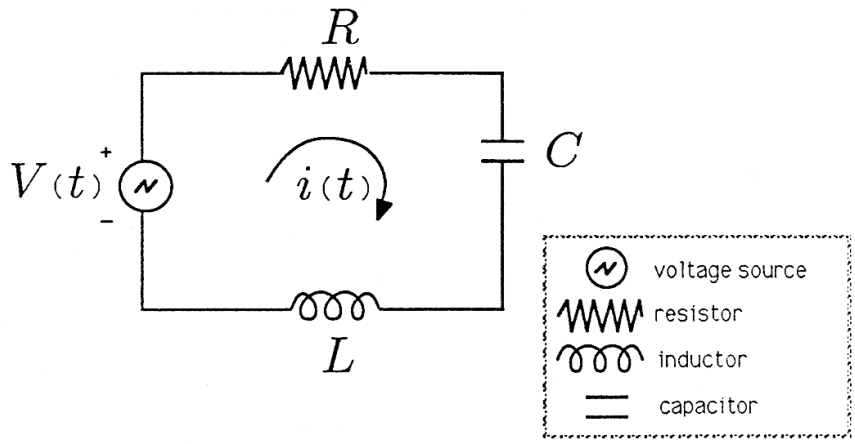
(MATLAB) Write a MATLAB program to compute and plot the Lissajous figure $z(t)$ when $A_x = 1/2$, $A_y = 1$, $\varphi_x = 0$, and $\varphi_y = \pi/6$. Discretize t appropriately and choose an appropriate range of values for t.

Phasors: Sinusoidal Steady State and the Series RLC Circuit

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Phasors may be used to analyze the behavior of electrical and mechanical systems that have reached a kind of equilibrium called *sinusoidal steady state*. In the sinusoidal steady state, every voltage and current (or force and velocity) in a system is sinusoidal with angular frequency ω. However, the amplitudes and phases of these sinusoidal voltages and currents are all different. For example, the voltage across a resistor might lead the voltage across a capacitor by 90° ($\frac{\pi}{2}$ radians) and lag the voltage across an inductor by 90° $\left(\frac{\pi}{2} radians\right)$.

In order to make our application of phasors to electrical systems concrete, we consider the series *RLC* circuit illustrated in [link]. The arrow labeled $i(t)$ denotes a current that flows in response to the voltage applied,and the + and - on the voltage source indicate that the polarity of the applied voltage is positive on the top and negative on the bottom. Our convention is that current flows from positive to negative, in this case clockwise in the circuit.



Series *RLC* Circuit

We will assume that the voltage source is an audio oscillator that pro- duces the voltage
Equation:

$$V(t) = A\cos(\omega t + \varphi).$$

We represent this voltage as the complex signal
Equation:

$$V(t) \leftrightarrow Ae^{j\varphi}e^{j\omega t}$$

and give it the phasor representation
Equation:

$$V(t) \leftrightarrow V; V = Ae^{j\varphi}.$$

We then describe the voltage source by the phasor *V* and remember that we can always compute the actual voltage by multiplying by $e^{j\omega t}$ and taking the real part:

Equation:

$$V(t) = \mathrm{Re}\left\{Ve^{j\omega t}\right\}.$$

Exercise:

Problem: Show that $\mathrm{Re}\left[Ve^{j\omega t}\right] = A\ \cos\ (\omega t + \varphi)$ when $V = Ae^{j\varphi}$.

Circuit Laws. In your circuits classes you will study the Kirchhoff laws that govern the low frequency behavior of circuits built from resistors (R), inductors (L), and capacitors (C). In your study you will learn that the voltage dropped across a resistor is related to the current that flows through it by the equation
Equation:

$$V_R(t) = Ri(t).$$

You will learn that the voltage dropped across an inductor is proportional to the derivative of the current that flows through it, and the voltage dropped across a capacitor is proportional to the integral of the current that flows through it:
Equation:

$$V_L(t) = L\frac{di}{dt}(t)$$
$$V_C(t) = \tfrac{1}{C}\int i(t)dt.$$

Phasors and Complex Impedance. Now suppose that the current in the preceding equations is sinusoidal, of the form
Equation:

$$i(t) = B\ \cos\ (\omega t + \theta).$$

We may rewrite $i(t)$ as

Equation:

$$i\left(t\right)=\mathrm{Re}\left\{Ie^{j\omega t}\right\}$$

where I is the phasor representation of $i(t)$.
Exercise:

Problem: Find the phasor I in terms of B and θ in [link].

The voltage dropped across the resistor is
Equation:

$$\begin{aligned} V_R\left(t\right) &= Ri\left(t\right) \\ &= R\,\mathrm{Re}\left\{Ie^{j\omega t}\right\} \\ &= \mathrm{Re}\left\{RIe^{j\omega t}\right\}. \end{aligned}$$

Thus the phasor representation for $V_R\left(t\right)$ is
Equation:

$$V_R\left(t\right)\leftrightarrow V_R; V_R=RI.$$

We call *R* the *impedance* of the resistor because *R* is the scale constant that relates the "phasor voltage V_R' to the "phasor current *I*."

The voltage dropped across the inductor is
Equation:

$$V_L\left(t\right)=L\frac{di}{dt}(t)=L\frac{d}{dt}\mathrm{Re}\left\{Ie^{j\omega t}\right\}.$$

The derivative may be moved through the $\mathrm{Re}[\ ]$ operator (see [link]) to produce the result
Equation:

$$\begin{aligned} V_L(t) &= L\,\mathrm{Re}\left\{j\omega I e^{j\omega t}\right\} \\ &= \mathrm{Re}\left\{j\omega L I e^{j\omega t}\right\}. \end{aligned}$$

Thus the phasor representation of $V_L(t)$
Equation:

$$V_L(t) \leftrightarrow V_L; V_L = j\omega L I.$$

We call $j\omega L$ the impedance of the inductor because $j\omega L$ is the complex scale constant that relates "phasor voltage V_L' to "phasor current I."
Exercise:

Problem: Prove that the operators $\dfrac{d}{dt}$ and $\mathrm{Re}[]$ commute:
Equation:

$$\frac{d}{dt}\mathrm{Re}\left\{e^{j\omega t}\right\} = \mathrm{Re}\left\{\frac{d}{dt}e^{j\omega t}\right\}.$$

The voltage dropped across the capacitor is
Equation:

$$V_C(t) = \frac{1}{C}\int i(t)dt = \frac{1}{C}\int \mathrm{Re}\left\{I e^{j\omega t}\right\}dt.$$

The integral may be moved through the $\mathrm{Re}[\ ]$ operator to produce the result
Equation:

$$\begin{aligned} V_C(t) &= \tfrac{1}{C}\mathrm{Re}\left\{\tfrac{I}{j\omega}e^{j\omega t}\right\} \\ &= \mathrm{Re}\left\{\tfrac{I}{j\omega C}e^{j\omega t}\right\}. \end{aligned}$$

Thus the phasor representation of $V_C(t)$ is
Equation:

$$V_C\left(t\right) \leftrightarrow V_C; V_C = \frac{I}{j\omega C}$$

We call $\frac{1}{j\omega C}$ the *impedance* of the capacitor because $\frac{1}{j\omega C}$ is the complex scale constant that relates “phasor voltage V_C" to “phasor current I.”

Kirchhoff's Voltage Law. Kirchhoff's voltage law says that the voltage dropped in the series combination of R, L, and C illustrated in [link] equals the voltage generated by the source (this is one of two fundamental conservation laws in circuit theory, the other being a conservation law for current):
Equation:

$$V\left(t\right) = V_R\left(t\right) + V_L\left(t\right) + V_C\left(t\right).$$

If we replace all of these voltages by their complex representations, we have
Equation:

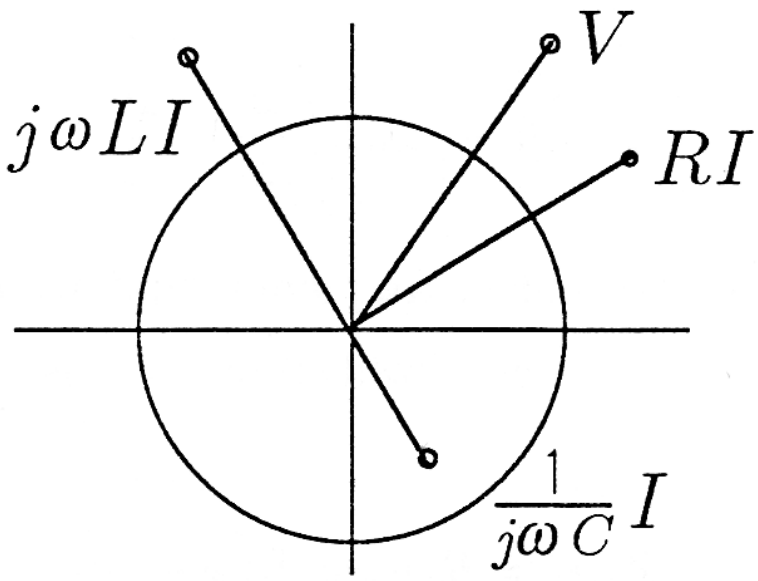
$$\mathrm{Re}\left\{Ve^{j\omega t}\right\} = \mathrm{Re}\left\{\left(V_R + V_L + V_C\right)e^{j\omega t}\right\}.$$

An obvious solution is
Equation:

$$\begin{aligned} V &= V_R + V_L + V_C \\ &= \left(R + j\omega L + \tfrac{1}{j\omega C}\right)I \end{aligned}$$

where *I* is the phasor representation for the current that flows in the circuit. This solution is illustrated in [link], where the phasor voltages RI, $j\omega LI$,

and $\frac{1}{j\omega C} I$ are forced to add up to the phasor voltage V.



Phasor Addition to Satisfy Kirchhoff's Law

Exercise:

Problem: Redraw [link] for $R = \omega L = \frac{1}{\omega C} = 1$.

Impedance. We call the complex number $R + j\omega L + \frac{1}{j\omega C}$ the complex impedance for the series *RLC* network because it is the complex number that relates the phasor voltage V to the phasor current I:
Equation:

$$V = ZI$$
$$Z = R + j\omega L + \tfrac{1}{j\omega C}.$$

The complex number Z depends on the numerical values of resistance (R), inductance (L), and capacitance (C), but it also depends on the angular frequency (ω) used for the sinusoidal source. This impedance may be manipulated as follows to put it into an illuminating form:
Equation:

$$\begin{aligned} Z &= R + j\left(\omega L - \tfrac{1}{\omega C}\right) \\ &= R + j\sqrt{\tfrac{L}{C}}\left(\omega\sqrt{LC} - \tfrac{1}{\omega\sqrt{LC}}\right). \end{aligned}$$

The parameter $\omega_0 = \frac{1}{\sqrt{LC}}$ is a parameter that you will learn to call an "undamped natural frequency" in your more advanced circuits courses. With it, we may write the impedance as
Equation:

$$Z = R + j\omega_0 L\left(\frac{\omega}{\omega_0} - \frac{\omega_0}{\omega}\right).$$

The frequency $\frac{\omega}{\omega_0}$ is a normalized frequency that we denote by ν. Then the impedence, as a function of normalized frequency, is
Equation:

$$Z(\nu) = R + j\omega_0 L\left(\nu - \frac{1}{\nu}\right).$$

When the normalized frequency equals one $(\nu = 1)$, then the impedance is entirely real and $Z = R$. The circuit looks like it is a single resistor.
Equation:

$$\begin{aligned} Z(\nu) &= R\left[1 + \left(\tfrac{\omega_0 L}{R}\right)^2\left(\nu - \tfrac{1}{\nu}\right)^2\right]^{1/2}. \\ \arg Z(\nu) &= \tan^{-1}\frac{\omega_0 L}{R}\left(\nu - \frac{1}{\nu}\right). \end{aligned}$$

The impedance obeys the following symmetries around $\nu = 1$:
Equation:

$$\begin{aligned} Z(\nu) &= Z^*\left(\tfrac{1}{\nu}\right) \\ Z(\nu) &= Z\left(\tfrac{1}{\nu}\right) \\ arg\ Z(\nu) &= -arg\ Z\left(\tfrac{1}{\nu}\right). \end{aligned}$$

In the next paragraph we show how this impedance function influences the current that flows in the circuit.

Resonance. The phasor representation for the current that flows the current that flows in the series *RLC* circuit is
Equation:

$$\begin{aligned} I &= \frac{V}{Z(\nu)} \\ &= \frac{1}{|Z(\nu)|} e^{-j\arg Z(\nu)} V \end{aligned}$$

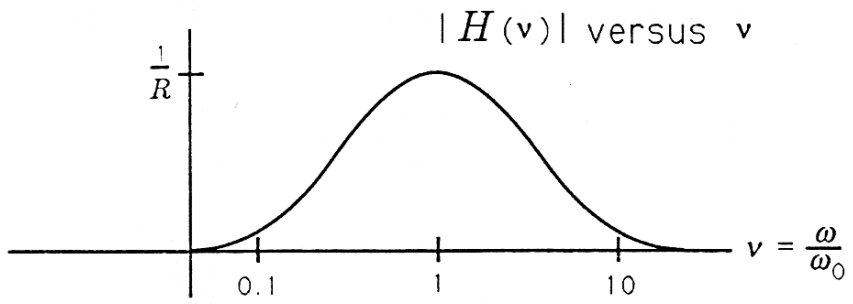
The function $H\left(\nu\right) = \frac{1}{Z(\nu)}$ displays a "resonance phenomenon." that is, $|H(\nu)|$ peaks at $\nu = 1$ and decreases to zero and $\nu = 0$ and $\nu = \infty$:
Equation:

$$|H(\nu)| = \begin{matrix} 0, & \nu = 0 \\ \frac{1}{R} & \nu = 1 \\ 0, & \nu = \infty. \end{matrix}$$

When $|H(\nu)| = 0$, no current flows.

The function $|H(\nu)|$ is plotted against the normalized frequency $\nu = \dfrac{\omega}{\omega_0}$ in Figure 3.14. The resonance peak occurs at $\nu = 1$, where $|H(\nu)| = \dfrac{1}{R}$

meaning that the circuit looks purely resistive. Resonance phenomena underlie the frequency selectivity of all electrical and mechanical networks.



Resonance in a Series *RLC* Circuit

Exercise:

Problem:

(MATLAB) Write a MATLAB program to compute and plot $|H(\nu)|$ and arg $H(\nu)$ versus ν for ν ranging from 0.1 to 10 in steps of 0.1. Carry out your computations for $\omega_0 \dfrac{L}{R} = 10, 1, 0.1$, and 0.01, and overplot your results.

Circle Criterion and Power Factor. Our study of the impedance $Z(\nu)$ and the function $H(\nu) = \dfrac{1}{Z(\nu)}$ brings insight into the resonance of an *RLC* circuit and illustrates the ffequency selectivity of the circuit. But there is more that we can do to illuminate the behavior of the circuit.

Equation:

$$V = RI + j\left(\omega L - \frac{1}{\omega C}\right) I.$$

This equation shows how voltage is divided between resistor voltage *RI* and inductor-capacitor voltage $j\left(\omega L - \frac{1}{\omega C}\right) I$.

Equation:

$$V = RI + j\omega_0 L\left(\frac{\omega}{\omega_0} - \frac{\omega_0}{\omega}\right)I$$

or
Equation:

$$V = RI + \frac{j\omega_0 L}{R}\left(\nu - \frac{1}{\nu}\right)RI.$$

In order to simplify our notation, we can write this equation as
Equation:

$$V = V_R + jk\left(\nu\right)V_R$$

where V_R is the phasor voltage RI and $k(\nu)$ is the real variable
Equation:

$$k\left(\nu\right) = \frac{\omega_0 L}{R}\left(\nu - \frac{1}{\nu}\right).$$

[link] brings very important geometrical insights. First, even though the phasor voltage V_R in the *RLC* circuit is complex, the terms V_R and $jk\left(\nu\right)V_R$ are out of phase by $\frac{\pi}{2}$ radians. This means that, for every allowable value of V_R, the corresponding $jk\left(\nu\right)V_R$ must add in a right triangle to produce the source voltage V. This is illustrated in [link]. As the frequency ν changes, then $k(\nu)$ changes, producing other values of V_R and $jk\left(\nu\right)V_R$ that sum to V. Several such solutions for V_R and $jk\left(\nu\right)V_R$ are illustrated in Figure 3.15(b). From the figure we gain the clear impression that the phasor voltage $V_{>\mathrm{R}}$ lies on a circle of radius $\frac{V}{2}$ centered at $\frac{V}{2}$
Let's try this solution,

Equation:

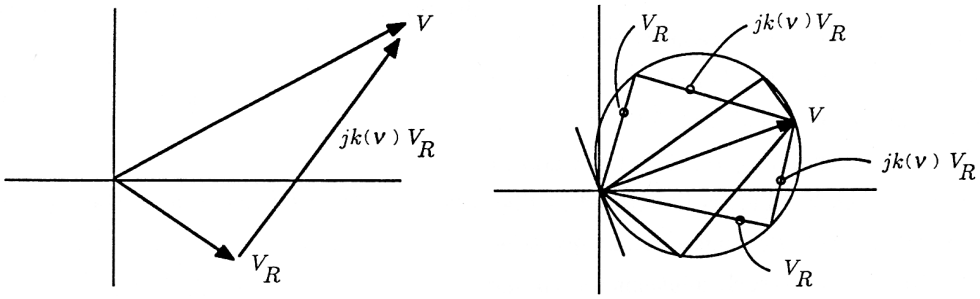
$$\begin{aligned} V_R &= \tfrac{V}{2} + \tfrac{V}{2} e^{j\psi} \\ &= \tfrac{V}{2}\left(1 + e^{j\psi}\right), \end{aligned}$$

and explore its consequences. When this solution is substituted into [link], the result is
Equation:

$$V = \frac{V}{2}\left(1 + e^{j\psi}\right) + jk\,(\nu)\frac{V}{2}\left(1 + e^{j\psi}\right)$$

or
Equation:

$$2 = \left(1 + e^{j\psi}\right)\left[1 + jk\,(\nu)\right].$$



The Components of V; (a) Addition of V_R and $jk\,(\nu)V_R$ to Produce V, and (b) Several Values of V_R and $jk\,(\nu)V_R$ that Produce V

If we multiply the left-hand side by its complex conjugate and the right-hand side by its complex conjugate, we obtain the identity
Equation:

$$4 = 2\,(1+\cos\psi)\left[1 + k^2\,(\nu)\right].$$

This equation tells us how the angle ψ depends on $k(\nu)$ and, conversely, how $k(\nu)$ depends on ψ:

Equation:

$$\cos\ \ \psi = \frac{1 - k^2\,(\nu)}{1 + k^2\,(\nu)}$$

Equation:

$$k^2\,(\nu) = \frac{1-\cos\psi}{1+\cos\psi}$$

The number $\cos\psi$ lies between -1 and $+1$, so a circular solution does indeed work.

Exercise:

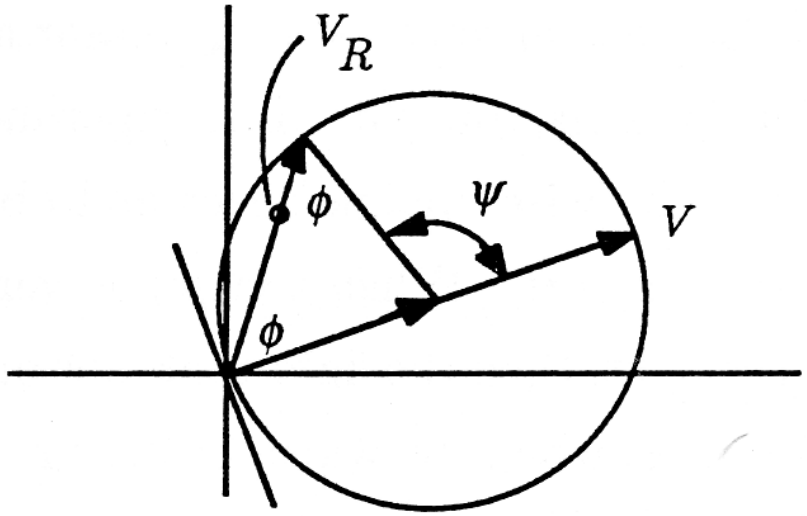
Problem:

Check $-1 \leq \cos\psi \leq 1$ for $-\infty < k < \infty$ and $-\infty < k < \infty$ for $-\pi \leq \psi \leq \pi$. Sketch k versus ψ and ψ versus k.

The equation $V_R = \dfrac{V}{2}\left(1 + e^{j\psi}\right)$ is illustrated in [link]. The angle that V_R makes with V is determined from the equation

Equation:

$$2\varphi + \pi - \psi = \pi \Rightarrow \varphi = \frac{\psi}{2}$$



The Voltages V and V_R, and the Power Factor $\cos \varphi$

In the study of power systems, $\cos \varphi$ is a "power factor" that determines how much power is delivered to the resistor. We may denote the power factor as

Equation:

$$\eta = \cos \varphi = \cos \frac{\psi}{2}.$$

But $\cos \psi$ may be written as

Equation:

$$\eta = \cos \varphi = \cos \frac{\psi}{2}$$

But $\cos \psi$ may be written as

Equation:

$$\begin{aligned}\cos\psi = \cos(\varphi+\varphi) &= \cos^2\varphi - -\sin^2\varphi \\ &= \cos^2\varphi - \left(1 - \cos^2\varphi\right) \\ &= 2\cos^2\varphi - 1 \\ &= 2\eta^2 - 1.\end{aligned}$$

Therefore the square of the power factor η is
Equation:

$$\eta^2 = \frac{\cos\psi + 1}{2} = \frac{1}{1 + k^2(\nu)}$$

The power factor is a maximum of 1 for $k(\nu) = 0$, corresponding to $\nu = 1$ $(\omega = \omega_0)$. It is a minimum of 0 for $k(\nu) = \pm\infty$, corresponding to $\nu = 0, \infty (\omega = 0, \infty)$.
Exercise:

Problem:

With k defined as $k(\nu) = \frac{\omega_0 L}{R}\left(\nu - \frac{1}{\nu}\right)$, plot $k^2(\nu)$, $\cos\psi$, and η^2 versus ν.

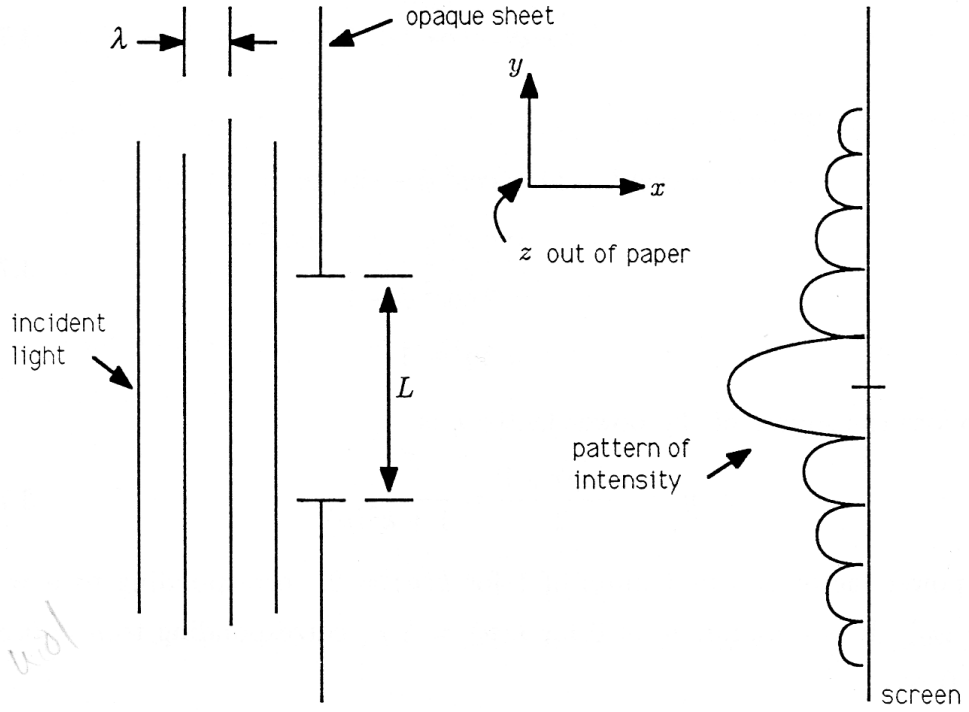
Exercise:

Problem: Find the value of ν that makes the power factor $\eta = 0.707$.

Phasors: Light Scattering by a Slit

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

One of the most spectacular successes for phasor analysis arises in the study of light diffraction by a narrow slit. The experiment is to shine laser light through a slit in an otherwise opaque sheet and observe the pattern of light that falls on a distant screen. When the slit is very narrow compared with the wavelength of the light, then the light that falls on the screen is nearly uniform in intensity. However, when the width of the slit is comparable to the wavelength of the light, the pattern of light that falls on the screen is scalloped in intensity, showing alternating light and dark bands. The experiment, and the observed results, are illustrated in [link].



Light Diffraction by a Slit

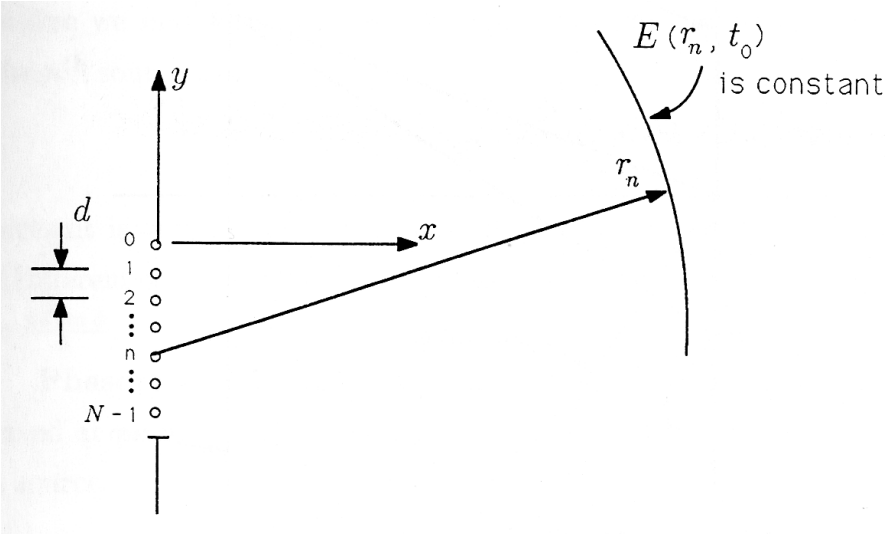
Why should this experiment produce this result? Phasor analysis illuminates the question and produces an elegant mathematical description of a profoundly important optical experiment.

Huygens's Principle. We will assume, as Christiaan Huygens did, that the light incident on the slit sets up a light field in the slit that may be modeled by N discrete sources, each of which radiates a "spherical wave of light." This model is illustrated in [link]. The distance between sources is d, and $Nd = L$ is the width of the slit. Each source is indexed by n, and n runs from 0 to $N - 1$. The 0th source is located at the origin of our coordinate system.

The spherical wave radiated by the nth source is described by the equation
Equation:

$$E\left(r_n, t\right) = \text{Re}\left\{\frac{A}{N} e^{j[\omega t - (2\pi/\lambda) r_n]}\right\}.$$

The function $E(r_n, t)$ describes the "electric field" at time *t* and distance r_n from the nth source. The field is constant as long as the variable $\omega t - (2\pi/\lambda) r_n$ is constant. Therefore, if we freeze time at $t = t_0$, the field will be constant on a sphere of radius r_n. This is illustrated in [link].



Huygens's Model for Light Diffraction

Exercise:

Problem:

Fix r_n in $E(r_n, t)$ and show that $E(r_n, t)$ varies cosinusoidally with time t. Sketch the function and interpret it. What is its period?

Exercise:

Problem:

Fix t in $E(r_n, t)$ and show that $E(r_n, t)$ varies cosinusoidally with radius r_n. Sketch the function and interpret it. Call the “period in r_n" the “wavelength.” Show that the wavelength is λ.

Exercise:

Problem:

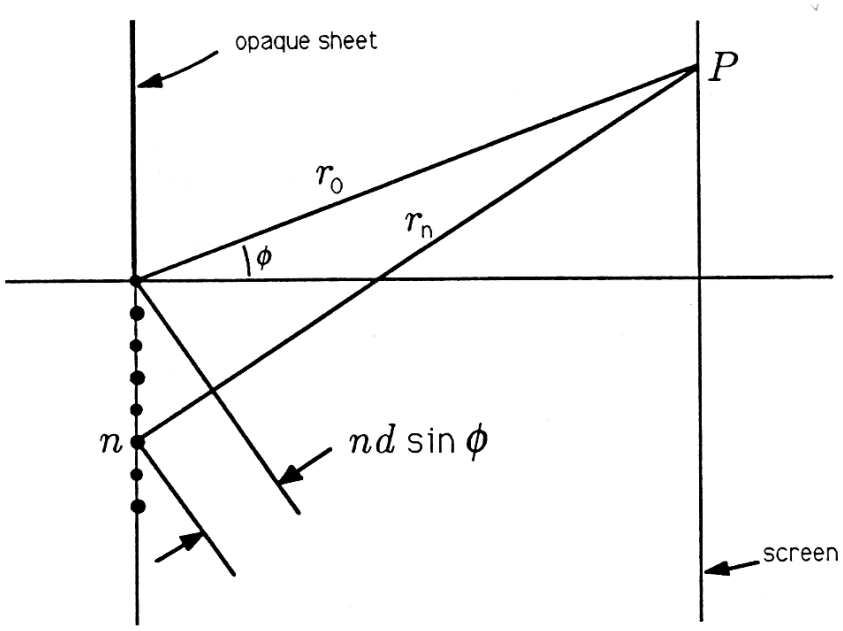
The “crest of the wave $E(r_n, t)$" occurs when $\omega t - (2\pi/\lambda) r_n = 0$. Show that the crest moves through space at velocity $v = \omega\lambda/2\pi$.

Geometry. If we now pick a point P on a distant screen, that point will be at distance r_0 from source 0, ..., r_n from source n, ..., and so on. If we

isolate the sources 0 and n, then we have the geometric picture of [link]. The angle φ is the angle that point P makes with the horizontal axis. The Pythagorean theorem says that the connection between distances r_0 and r_n is

Equation:

$$(r_n - nd \sin \varphi)^2 + (nd \cos \varphi)^2 = r_0^2.$$



Geometry of the Experiment

Let's try the solution

Equation:

$$r_n = r_0 + nd \sin \varphi.$$

This solution produces the approximate identity

Equation:

$$r_0^2 + (nd \cos \varphi)^2 \cong r_0^2$$

Equation:

$$r_0^2 + (nd\cos\varphi)^2 \cong r_0^2$$
$$1 + \left(\frac{nd}{r_0}\ cos\ \varphi\right)^2 \cong 1.$$

This will be close for $\frac{nd}{r_0} << 1$. We will assume that the slit width L is small compared to the distance to any point on the screen. Then $\frac{Nd}{r_0} = \frac{L}{r_0} << 1$, in which case the approximate solution for r_n is valid for all n. This means that, for any point P on the distant screen, the light contributed by the nth source is approximately
Equation:

$$\begin{aligned} E_n(\varphi, t) &= \operatorname{Re}\left\{\tfrac{A}{N} e^{j[\omega t - (2\pi/\lambda)(r_0 + nd\sin\varphi)]}\right\} \\ &= \operatorname{Re}\left\{\tfrac{A}{N} e^{-j(2\pi/\lambda)r_0} e^{-j(2\pi/\lambda)nd\sin\varphi} e^{j\omega t}\right\}. \end{aligned}$$

The phasor representation for this function is just
Equation:

$$E_n(\varphi) = \frac{A}{N} e^{-j(2\pi/\lambda)r_0} e^{-j(2\pi/\lambda)nd\sin\varphi}.$$

Note that $E_0(\varphi)$, the phasor associated with the 0th source, is $\frac{A}{N} e^{-j(2\pi/\lambda)r_0}$. Therefore we may write the phasor representation for the light contributed by the nth source to be
Equation:

$$E_n(\varphi) = E_0(\varphi) e^{-j(2\pi/\lambda)nd\sin\varphi}.$$

This result is very important because it shows the light arriving at point P from different sources to be “out of phase” by an amount that depends on the ratio $\frac{nd \sin \varphi}{\lambda}$

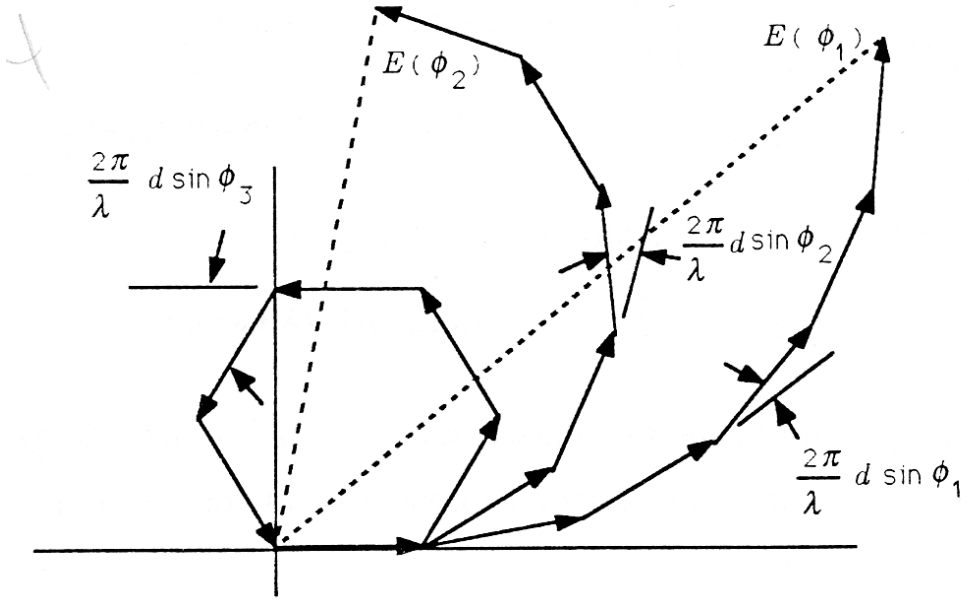
Phasors and Interference. The phasor representation for the field observed at point P on the screen is the sum of the phasors contributed by each source:
Equation:

$$E(\varphi) = \sum_{n=0}^{N-1} E_n(\varphi) = E_0(\varphi) \sum_{n=0}^{N-1} e^{-j(2\pi/\lambda)nd\sin\varphi}.$$

This is a sum of the form
Equation:

$$E(\varphi) = E_0(\varphi) \sum_{n=0}^{N-1} e^{jn\theta}$$

where the angle θ is $(2\pi/\lambda)d \sin \varphi$. This sum is illustrated in [link] for several representative values of θ. Note that for small θ, meaning small φ, the sum has large magnitude, whereas for θ on the order of $2\pi/N$, the sum is small. This simple geometric interpretation shows that for some values of φ, corresponding to some points P on the screen, there will be constructive interference between the phasors, while for other values of φ there will be destructive interference. Constructive interference produces bright light, and destructive interference produces darkness.



Phasor Sums for Diffraction

The geometry of [link] is illuminating. However, we already know from our study of complex numbers and geometric sums that the phasor sum of Equation 3.85 may be written as
Equation:

$$E(\varphi) = E_0(\varphi)\frac{1 - e^{-j(2\pi/\lambda)Nd\sin\varphi}}{1 - e^{-j(2\pi/\lambda)d\sin\varphi}}.$$

This result may be manipulated to produce the form
Equation:

$$E(\varphi) = \frac{A}{N}e^{-j(2\pi/\lambda)r_0}e^{-j(\pi/\lambda)(N-1)d\sin\varphi}\frac{\sin\left(N\pi\frac{d}{\lambda}\sin\varphi\right)}{\sin\left(\pi\frac{d}{\lambda}\sin\varphi\right)}$$

The magnitude is the intensity of the light at angle φ from horizontal:
Equation:

$$|E(\varphi)| = |\frac{A}{N} \frac{\sin\left(N\pi\frac{d}{\lambda}\sin\varphi\right)}{\sin\left(\pi\frac{d}{\lambda}\sin\varphi\right)}|.$$

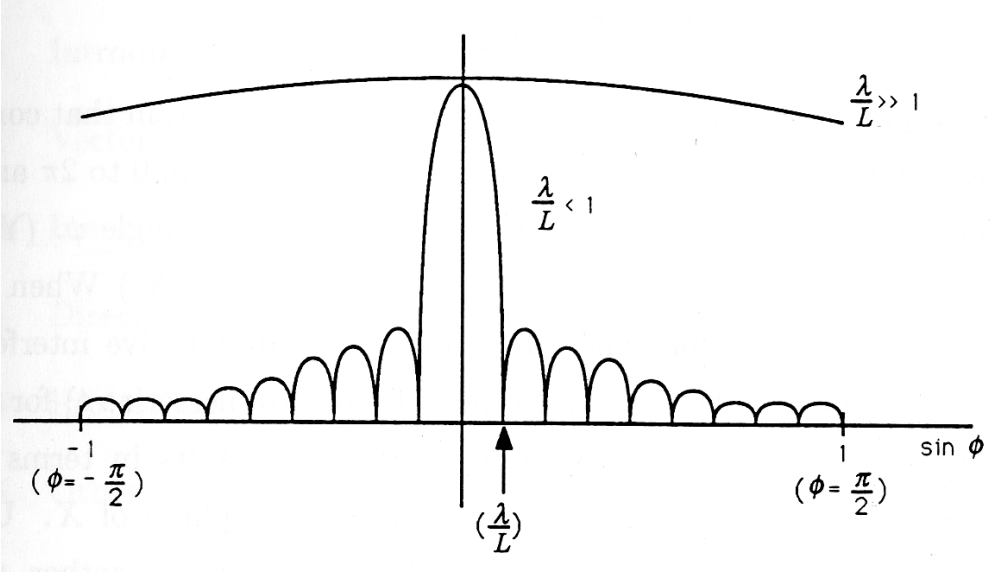
Exercise:

Problem: Derive [link] from [link].

Limiting Form. Huygens's model is exact when d shrinks to 0 and N increases to infinity in such a way that $Nd \to L$, the slit width. Then
Equation:

$$|E(\varphi)| \to |\frac{A\sin\left(\frac{\pi L}{\lambda}\sin\varphi\right)}{\frac{\pi L}{\lambda}\sin\varphi}|.$$

This function is plotted in [link] for two values of $\frac{L}{\lambda}$ the width of the slit measured in wavelengths. When $\frac{L}{\lambda} << 1$ (i.e., $\frac{\lambda}{L} >> 1$), then the light is uniformly distributed on the screen. However, when $\frac{L}{\lambda} > 1\left(\frac{\lambda}{L} < 1\right)$, then the function has many zeros for $|\sin\varphi| < 1$, as illustrated in the figure. These zeros correspond to dark spots on the screen where the fields radiated from the infinity of points within the slit interfere destructively.



Interference Pattern for a Slit

Exercise:

Problem: Derive [link] from [link].

Exercise:

Problem: (MATLAB) Plot the discrete approximation
Equation:

$$\left| \frac{\frac{A}{N} \sin\left(N\pi \frac{d}{\lambda} \sin\varphi\right)}{\sin\left(\pi \frac{d}{\lambda} \sin\varphi\right)} \right|$$

versus $\sin\varphi$ for $\frac{L}{\lambda} = \frac{Nd}{\lambda} = 10$ and $N = 2, 4, 8, 16, 32$. Compare with the continuous, limiting form
Equation:

$$\left| \frac{A \sin\left(\pi \frac{L}{\lambda} \sin \varphi\right)}{\pi \frac{L}{\lambda} \sin \varphi} \right|$$

Phasors: Numerical Experiment (Interference Patterns)

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Let's add two sinusoids whose amplitudes and frequencies are identical and whose phases are different:
Equation:

$$x(t) = A \cos(\omega t + \varphi) + A \cos(\omega t + \varphi + \psi).$$

Show analytically that this sum has the phasor representation
Equation:

$$X = 2A\ cos\ \left(\frac{\psi}{2}\right) e^{j[\varphi + (\psi/2)]}.$$

Interpret this finding. Then write a MATLAB program that computes and plots complex X on the complex plane as ψ varies from 0 to 2π and that plots magnitude, $|X|$, and phase, $\arg X$, versus the phase angle ψ. (You will have to choose $\psi = n\frac{2\pi}{N}, n = 0, 1, ..., N-1$, for a suitable N.)
When do you get constructive interfelence and when do you get destructive interference? Now compute and plot $x(t)$ versus t (you will need to discretize t) for several interesting values of ψ. Explain your interference results in terms of the amplitude and phase of $x(t)$ and the magnitude and phase of X. Use the subplots discussed in "An Introduction to MATLAB" to plot all of your results together.

Linear Algebra: Introduction

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

We use this chapter to introduce students to the algebraic structure of *vectors* and *matrices* and to introduce them to matrix computations. These matrix computations are used in the chapters "Vector Graphics", "Filtering", and "Binary Codes" to solve problems in vector graphics, filtering, and binary coding.

Vectors are introduced in the section "Vectors", along with algebraic and geometric interpretations of some fundamental vector operations and properties. The section "Inner Product and Euclidean Norm", the section "Direction Cosines", and the section "Projections" introduce inner products and their applications, including norm, direction cosines, orthogonality, and projections. Some important alternatives to the Euclidean norm are introduced in the section "Other Norms". Matrices are motivated and introduced in this section. The notation in these sections can be daunting to the beginner, so we proceed very carefully, using example after example. In the section "Solving Linear Systems of Equations" we codify the elimination procedures that students have used in high school to solve linear systems of equations. The MATLAB demonstration in Demo 2 shows how to use MATLAB to solve linear equations. The section "Circuit Analysis" shows how linear algebra and MATLAB can be used to analyze dc circuits. The numerical experiment "Circuit Design" gives students practice in building function files in MATLAB and shows how to solve a sequence of linear equations in order to *design* a circuit with desired properties.

Occasionally we have placed important results in the problems. We feel that students should not miss the material in Exercise 3 in "Vectors", Exercise 3

in "Inner Product and Euclidean Norm", Exercise 3 in "Projections", Exercise 1 in "Matrices", and Exercise 4 in "Matrices".

Introduction

Linear algebra is a branch of mathematics that is used by engineers and applied scientists to design and analyze complex systems. Civil engineers use linear algebra to design and analyze load-bearing structures such as bridges. Mechanical engineers use linear algebra to design and analyze suspension systems, and electrical engineers use it to design and analyze electrical circuits. Electrical, biomedical, and aerospace engineers use linear algebra to enhance X rays, tomographs, and images from space. In this chapter and the next we study two common problems from electrical engineering and use linear algebra to solve them. The two problems are (i) electrical circuit analysis and (ii) coordinate transformations for computer graphics. The first of these applications requires us to understand the solution of linear systems of equations, and the second requires us to understand the representation of mathematical operators with matrices.

Much of linear algebra is concerned with systematic techniques for organizing and solving simultaneous linear equations by elimination and substitution. The following example illustrates the basic ideas that we intend to develop.

Example:
A woman steps onto a moving sidewalk at a large airport and stands while the moving sidewalk moves her forward at 1.2 meters/seconds. At the same time, a man begins walking against the motion of the sidewalk from the opposite end at 1.5 meters/second (relative to the sidewalk). If the moving sidewalk is 85 meters long, how far does each person travel (relative to the ground) before they pass each other?
To solve this problem, we first assign a variable to each unknown quantity. Let x_1 be the distance traveled by the woman, and let x_2 be the distance traveled by the man. The sum of the two distances is 85 meters, giving us one equation:

Equation:

$$x_1 + x_2 = 85.$$

Our second equation is based on the time required before they pass. Time equals distance divided by rate, and the time is the same for both people:
Equation:

$$\frac{x_1}{1.2} = \frac{x_2}{1.5 - 1.2} \Rightarrow 0.3x_1 - 1.2x_2 = 0.$$

We may substitute [link] into [link] to obtain the result $\frac{1.2}{0.3}x_2 + x_2 = 85$, or
Equation:

$$5x_2 = 85 \Rightarrow x_2 = 17.$$

Combining the result from [link] with that of [link], we find that
Equation:

$$x_1 = 68.$$

So the man travels 17 meters, and the woman travels 68 meters.
[link] and [link] are the key equations of [link]. They may be organized into the "matrix equation"
Equation:

$$\begin{bmatrix} 1 & 1 \\ 0.3 & -1.2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 85 \\ 0 \end{bmatrix}.$$

The rules for matrix-vector multiplication are evidently
Equation:

$$(1)\ x_1 + (1)x_2 = 85$$

Equation:

$$(0.3)x_1 + (-1.2)x_2 = 0.$$

[link] and [link] may be organized into the matrix equation

Equation:

$$\begin{bmatrix} 0 & 5 \\ 0.3 & -1.2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 85 \\ 0 \end{bmatrix}.$$

This equation represents one partially solved form of [link], wherein we have used the so-called *Gauss elimination procedure* to introduce a zero into the matrix equation in order to isolate one variable. The MATLAB software contains built-in procedures to implement Gauss elimination on much larger matrices. Thus MATLAB may be used to solve large systems of linear equations.

Before we can apply linear algebra to more interesting physical problems, we need to introduce the mathematical tools we will use.

Linear Algebra: Vectors

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

For our purposes, a *vector* is a collection of *real numbers* in a one-dimensional array.[footnote] We usually think of the array as being arranged in a column and write
In a formal development of linear algebra, the abstract concept of a vector space plays a fundamental role. We will leave such concepts to a complete course in linear algebra and introduce only the functional techniques necessary to solve the problems at hand.

$$\mathrm{x} = \begin{matrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{matrix} .$$

Notice that we indicate a vector with boldface and the constituent elements with subscripts. A real number by itself is called a *scalar*, in distinction from a vector or a matrix. We say that x is an n-vector, meaning that x has n elements. To indicate that x_1 is a real number, we write
Equation:

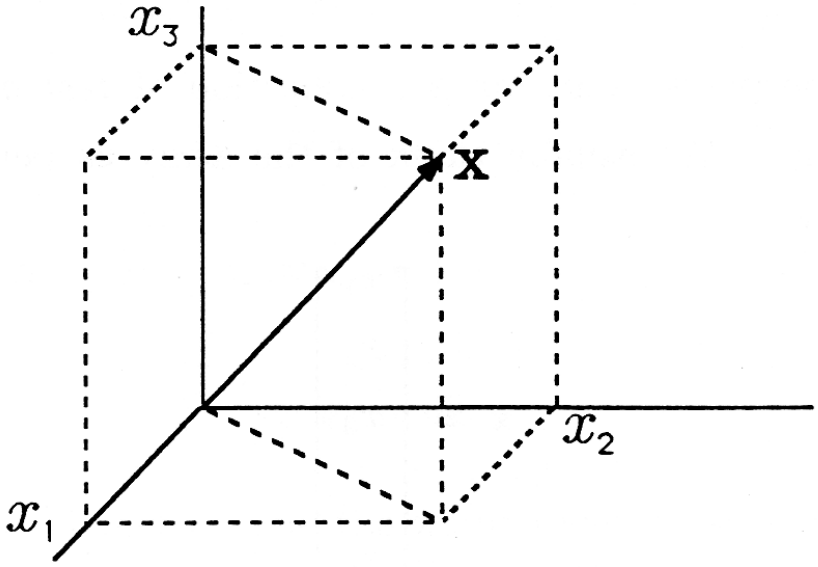
$$x_1 \in \quad ,$$

meaning that x_1 is contained in , the set of real numbers. To indicate that x is a vector of n real numbers, we write

Equation:

$$\mathbf{x} \in \quad ^{n},$$

meaning that x is contained in R^n, the set of real n-tuples. Geometrically, $R^n$is n-dimensional space, and the notation $\mathbf{x} \in \quad ^{n}$ means that x is a point in that space, specified by the n coordinates $x_1, x_2, \ldots, x_n$. [link] shows a vector in R^3, drawn as an arrow from the origin to the point x. Our geometric intuition begins to fail above three dimensions, but the linear algebra is completely general.



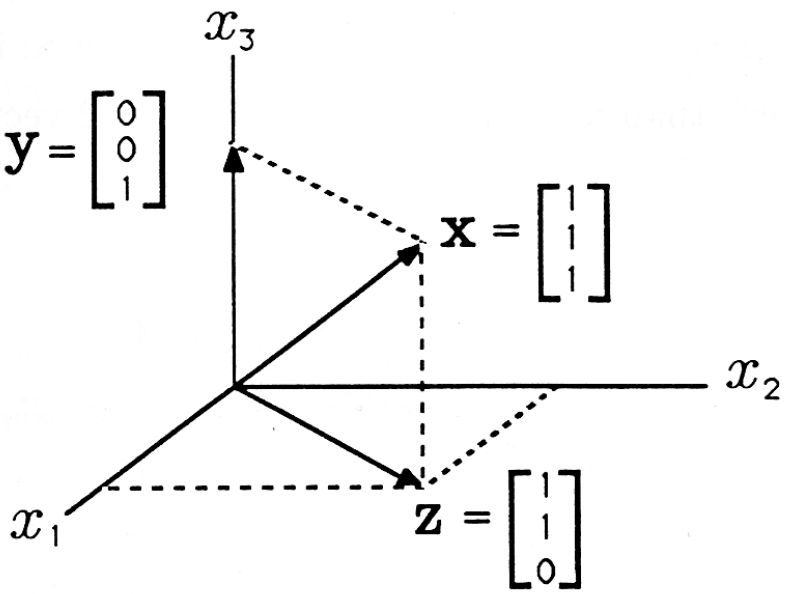
A Vector in R^3

We sometimes find it useful to sketch vectors with more than three dimensions in the same way as the three-dimensional vector of [link]. We then consider each axis to represent more than one dimension, a hyperplane, in our n-dimensional space. We cannot show all the details of what is happening in n-space on a three-dimensional figure, but we can often show important features and gain geometrical insight.

Vector Addition. Vectors with the same number of elements can be added and subtracted in a very natural way:
Equation:

$$x + \mathrm{y} = \begin{matrix} x_1 + y_1 \\ x_2 + y_2 \\ x_3 + y_3 \\ | \\ x_n + y_n \end{matrix} ; x - y = \begin{matrix} x_1 - y_1 \\ x_2 - y_2 \\ x_3 - y_3 \\ | \\ x_n - y_n \end{matrix} .$$

Example:
The difference between the vector $\mathrm{x} = \begin{matrix} 1 \\ 1 \\ 1 \end{matrix}$ and the vector $y = \begin{matrix} 0 \\ 0 \\ 1 \end{matrix}$ is the vector $z = x - y = \begin{matrix} 1 \\ 1 \\ 0 \end{matrix}$. These vectors are illustrated in [link]. You can see that this result is consistent with the definition of vector subtraction in [link]. You can also picture the subtraction in [link] by mentally reversing the direction of vector y to get $-y$ and then adding it to x by sliding it to the position where its tail coincides with the head of vector x. (The head is the end with the arrow.) When you slide a vector to a new position for adding to another vector, you must not change its length or direction.


x_3
$\mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
$\mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
x_2
x_1
$\mathbf{z} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$

Subtraction of Vectors

Exercise:

Problem:

Compute and plot $x + y$ and $x - y$ for each of the following cases:

$$\text{a. } \mathrm{x} = \begin{matrix} 1 \\ 3 \\ 2 \end{matrix}, \ \mathrm{y} = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix};$$

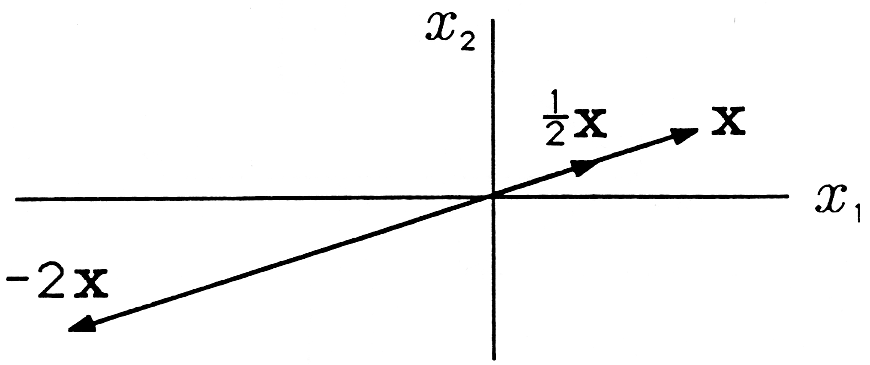
$$\text{b. } \mathrm{x} = \begin{matrix} -1 \\ 3 \\ -2 \end{matrix}, \ \mathrm{y} = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix};$$

$$\text{c. } \mathrm{x} = \begin{matrix} 1 \\ -3 \\ 2 \end{matrix}, \ \mathrm{y} = \begin{matrix} 1 \\ 3 \\ 2 \end{matrix}.$$

Scalar Product. Several different kinds of vector multiplication are defined.[footnote] We begin with the *scalar product*. Scalar multiplication is defined for scalar a and vector x as
The division of two vectors is undefined, although three different ‚Äúdivisions‚Äù are defined in MATLAB.
Equation:

$$ax = \begin{matrix} ax_1 \\ ax_2 \\ ax_3 \\ | \\ ax_n \end{matrix} .$$

If $|a| < 1$, then the vector ax is "shorter" than the vector x; if $|a| > 1$, then the vector ax is ‚"longer" than x. This is illustrated for a 2-vector in [link].



The Scalar Product ax

Exercise:

Problem:

Compute and plot the scalar product $a\mathrm{x}$ when $\mathrm{x} = \begin{matrix} 1 \\ 1/2 \\ 1/4 \end{matrix}$ for each of the following scalars:

a. $a = 1$;
b. $a = -1$;
c. $a = -1/4$;
d. $a = 2$.

Exercise:

Problem:

Given vectors , , $\in$ ^{n} and the scalar $a \in$, prove the following identities:

a. $\mathrm{x} + \mathrm{y} = \mathrm{y} + \mathrm{x}$. Is vector addition commutative?
b. $(\mathrm{x} + \mathrm{y}) + \mathrm{z} = \mathrm{x} + (\mathrm{y} + \mathrm{z})$. Is vector addition associative?
c. $a(\mathrm{x} + \mathrm{y}) = a\mathrm{x} + a\mathrm{y}$. Is scalar multiplication distributive over vector addition?

Linear Algebra: Inner Product and Euclidean Norm

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

The inner product (x, y) between vectors x and y is a scalar consisting of the following sum of products:
Equation:

$$(\mathrm{x}, \mathrm{y}) = x_1 y_1 + x_2 y_2 + x_3 y_3 + \cdots + x_n y_n.$$

This definition seems so arbitrary that we wonder what uses it could possibly have. We will show that the inner product has three main uses:

i. computing length or “norm”,
ii. finding angles between vectors and checking for “orthogonality”, and
iii. computing the “component of one vector along another” (projection).

Since the inner product is so useful, we need to know what algebraic operations are permitted when we are working with inner products. The following are some properties of the inner product. Given $\mathrm{x}, \mathrm{y}, \mathrm{z} \in \mathscr{R}^n$ and $a \in \mathscr{R}$,

i. $(\mathrm{x}, \mathrm{y}) = (\mathrm{y}, \mathrm{x})$;
ii. $(a\mathrm{x}, \mathrm{y}) = a(\mathrm{x}, \mathrm{y}) = (\mathrm{x}, a\mathrm{y})$; and
iii. $(\mathrm{x}, \mathrm{y} + \mathrm{z}) = (\mathrm{x}, \mathrm{y}) + (\mathrm{x}, \mathrm{z})$.

Exercise:

Problem:

Prove the three preceding properties by using the definition of inner product. Is the equation $\mathrm{x}(\mathrm{y}, \mathrm{z}) = (\mathrm{x}, \mathrm{y})\mathrm{z}$ also a true property? Prove or give a counterexample.

Euclidean Norm. Sometimes we want to measure the length of a vector, namely, the distance from the origin to the point specified by the vector's coordinates. A vector's length is called the *norm* of the vector. Recall from Euclidean geometry that the distance between two points is the square root of the sum of the squares of the distances in each dimension. Since we are measuring from the origin, this implies that the norm of the vector *x* is
Equation:

$$\| \mathrm{x} \| = \sqrt{x_1^2 + x_2^2 + + x_n^2}.$$

Notice the use of the double vertical bars to indicate the norm. An equivalent definition of the norm, and of the norm squared, can be given in terms of the inner product:
Equation:

$$\| \mathrm{x} \| = \sqrt{(\mathrm{x}, \mathrm{x})}$$

or
Equation:

$$\| \mathrm{x} \|^2 = (\mathrm{x}, \mathrm{x}).$$

Example:
The Euclidean norm of the vector
Equation:

$$\mathrm{x} = \begin{matrix} 1 \\ 3 \\ 5 \\ -2 \end{matrix},$$

is $\| \mathrm{x} \| = \sqrt{1^2 + 3^2 + 5^2 + (-2)^2} = \sqrt{39}.$

An important property of the norm and scalar product is that, for any $\mathrm{x} \in \mathscr{R}^n$ and $a \in \mathscr{R}$,

Equation:

$$||a\mathrm{x}|| = |a|||\mathrm{x}||.$$

So we can take a scalar multiplier outside of the norm if we take its absolute value.

Exercise:

Problem: Prove [link].

Cauchy-Schwarz Inequality. Inequalities can be useful engineering tools. They can often be used to find the best possible performance of a system, thereby telling you when to quit trying to make improvements (or proving to your boss that it can't be done any better). The most fundamental inequality in linear algebra is the Cauchy-Schwarz inequality. This inequality says that the inner product between two vectors x and y is less than or equal (in absolute value) to the norm of x times the norm of y, with equality if and only if $\mathrm{y} = \alpha\mathrm{x}$:

Equation:

$$|(\mathrm{x}, \mathrm{y})| \leq ||\mathrm{x}||||\mathrm{y}||.$$

To prove this theorem, we construct a third vector $\mathrm{z} = \lambda\mathrm{x} - \mathrm{y}$ and measure its norm squared:
Equation:

$$||\lambda\mathrm{x} - \mathrm{y}||^2 = (\lambda\mathrm{x} - \mathrm{y}, \lambda\mathrm{x} - \mathrm{y}) = \lambda^2||\mathrm{x}||^2 - 2\lambda\,(\mathrm{x}, \mathrm{y}) + ||\mathrm{y}||^2 \geq 0.$$

So we have a polynomial in λ that is always greater than or equal to 0 (because every norm squared is greater than or equal to 0). Let's assume that x and y are given and minimize this norm squared with respect to λ. To do this, we take the derivative with respect to λ and equate it to 0:
Equation:

$$2\lambda||\mathrm{x}||^2 - 2\,(\mathrm{x}, \mathrm{y}) = 0 \Rightarrow \lambda = \frac{(\mathrm{x}, \mathrm{y})}{||\mathrm{x}||^2}$$

When this solution is substituted into the formula for the norm squared in [link], we obtain
Equation:

$$\frac{(\mathrm{x}, \mathrm{y})}{||\mathrm{x}||^2}^2 ||\mathrm{x}||^2 - \frac{2(\mathrm{x}, \mathrm{y})}{||\mathrm{x}||^2}(\mathrm{x}, \mathrm{y}) + ||\mathrm{y}||^2 \geq 0,$$

which simplifies to
Equation:

$$-\frac{(\mathrm{x}, \mathrm{y})^2}{||\mathrm{x}||^2} + ||\mathrm{y}||^2 \geq 0 \Rightarrow (\mathrm{x}, \mathrm{y})^2 \leq ||\mathrm{x}||^2||\mathrm{y}||^2.$$

The proof of the Cauchy-Schwarz inequality is completed by taking the positive square root on both sides of [link]. When $\mathrm{y} = \alpha\mathrm{x}$, then
Equation:

$$
\begin{aligned}
(\mathrm{x},\mathrm{y})^2 = (\mathrm{x},\alpha\mathrm{x})^2 &= \left[|\alpha|\,(\mathrm{x},\mathrm{x})\right]^2 = \left(|\alpha|||\mathrm{x}||^2\right)^2 \\
&= (|\alpha|^2||\mathrm{x}||^2)||\mathrm{x}||^2 \\
&= (\alpha\mathrm{x},\alpha\mathrm{x})||\mathrm{x}||^2 \\
&= (\mathrm{y},\mathrm{y})||\mathrm{x}||^2 \\
&= ||\mathrm{y}||^2||\mathrm{x}||^2,
\end{aligned}
$$

which shows that equality holds in [link] when y is a scalar multiple of x.

Exercise:

Problem:

Use the Cauchy-Schwarz inequality to prove the triangle inequality, which states

Equation:

$$||\mathrm{x}+\mathrm{y}|| \leq ||\mathrm{x}|| + ||\mathrm{y}||.$$

Explain why this is called the *triangle inequality*.

Linear Algebra: Direction Cosines

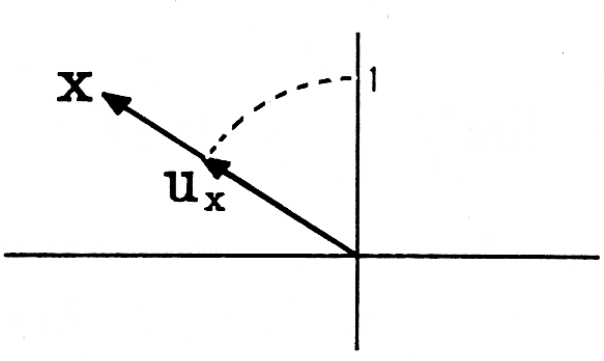
Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Unit Vectors. Corresponding to every vector x is a *unit vector* u_x pointing in the same direction as x. The term *unit vector* means that the norm of the vector is 1:
Equation:

$$||\mathrm{u_x}|| = 1.$$

The question is, given x, how can we find u_x? The first part of the answer is that u_x will have to be a positive scalar multiple of x in order to point in the same direction as x, as shown in [link]. Thus
Equation:

$$\mathrm{u_x} = \alpha \mathrm{x}.$$



The Unit Vector u_x

But what is α? We need to choose α so that the norm of u_x will be 1:
Equation:

$$||\mathrm{u_x}|| = 1$$

Equation:

$$|\alpha \mathrm{x}|| = 1$$

Equation:

$$|\alpha|||\mathrm{x}|| = 1$$

Equation:

$$\alpha = \frac{1}{||\mathrm{x}||}$$

We have dropped the absolute value bars on α because $||\mathrm{x}||$ is positive. The α that does the job is 1 over the norm of x. Now we can write formulas for u_x in terms of x and x in terms of u_x :
Equation:

$$\mathrm{u_x} = \frac{1}{||\mathrm{x}||}\mathrm{x},$$

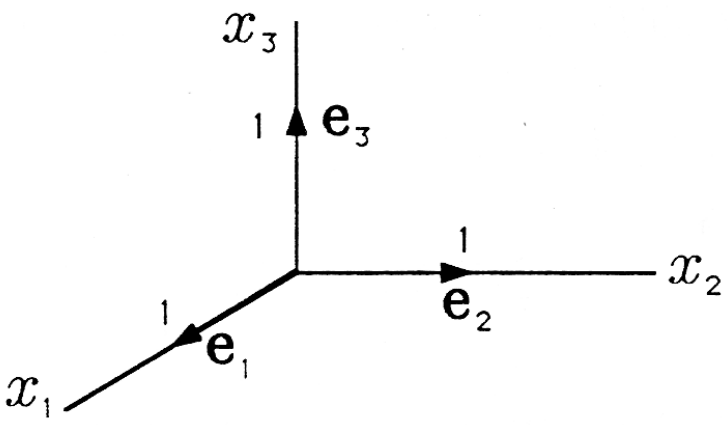
Equation:

$$\mathrm{x} = ||\mathrm{x}||\mathrm{u_x}.$$

So the vector x is its direction vector u_x, scaled by its Euclidean norm.

Unit Coordinate Vectors. There is a special set of unit vectors called the unit coordinate vectors. The *unit coordinate vector*e_i is a unit vector in ∇^n

that points in the positive direction of the ith coordinate axis. [link] shows the three unit coordinate vectors in ∇^3.



Unit Coordinate Vectors in R^3

For three-dimensional space,R^3, the unit coordinate vectors are
Equation:

$$e_1 = \begin{matrix}1\\0\\0\end{matrix}, \; e_2 = \begin{matrix}0\\1\\0\end{matrix}, \; e_3 = \begin{matrix}0\\0\\1\end{matrix}.$$

More generally, in n-dimensional space, there are n unit coordinate vectors given by

$$\mathbf{e}_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \longleftarrow \ i^{\text{th}} \text{ element} \\ \\ \\ \longleftarrow \ n \text{ elements.} \end{matrix}$$

You should satisfy yourself that this definition results in vectors with a norm of l.

Exercise:

Problem: Find the norm of the vector au where u is a unit vector.

Exercise:

Problem: Find the unit vector u_x in the direction of

a. $\mathrm{x} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$;

b. $\mathrm{x} = \begin{matrix} 3 \\ 6 \\ -2 \\ 1 \end{matrix}$;

c. $\mathrm{x} = \begin{matrix} -1 \\ 1 \\ -1 \end{matrix}$.

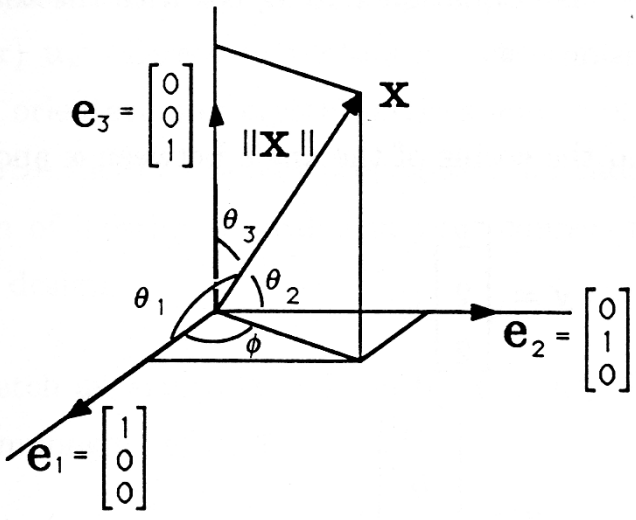
Direction Cosines. We often say that vectors “have magnitude and direction.” This is more or less obvious from "Linear Algebra: Vectors", where the three-dimensional vector x has length $\sqrt{x_1^2 + x_2^2 + x_3^2}$ and points in a direction defined by the components x_1, x_2, and x_3. It is perfectly obvious from [link] where x is written as $\mathrm{x} = ||\mathrm{x}||\mathrm{u}_\mathrm{x}$. But perhaps there is another representation for a vector that places the notion of magnitude and direction in even clearer evidence.

[link] shows an arbitrary vector $\mathrm{x} \in \mathscr{R}^3$ and the three-dimensional *unit coordinate vectors* $\mathrm{e}_1, \mathrm{e}_2, \mathrm{e}_3$. The inner product between the vector x and the unit vector e_k just reads out the kth component of x:
Equation:

$$(\mathrm{x}, \mathrm{e}_k) = (\mathrm{e}_k, \mathrm{x}) = x_k.$$

Since this is true even in R^n, any vector $\mathrm{x} \in \mathscr{R}^n$ has the following representation in terms of unit vectors:
Equation:

$$\mathrm{x} = (\mathrm{x}, \mathrm{e}_1)\mathrm{e}_1 + (\mathrm{x}, \mathrm{e}_2)\mathrm{e}_2 + \cdots + (\mathrm{x}, \mathrm{e}_n)\mathrm{e}_n.$$



The Three-Dimensional Unit Vectors

Let us now generalize our notion of an angle θ between two vectors to R^n as follows:
Equation:

$$\cos\,\theta = \frac{(\mathrm{x},\mathrm{y})}{||\mathrm{x}||||\mathrm{y}||}$$

The celebrated Cauchy-Schwarz inequality establishes that $-1- \leq \cos\,\theta \leq 1$. With this definition of angle, we may define the angle θ_k that a vector makes with the unit vector e_k to be
Equation:

$$\cos\,\theta_k = \frac{(\mathrm{x},\mathrm{e}_k)}{||\mathrm{x}||||\mathrm{e}_k\,||}$$

But the norm of e_k is 1, so
Equation:

$$\cos\,\theta_k = \frac{(\mathrm{x},\mathrm{e}_k)}{||\mathrm{x}||} = \frac{x_k}{||\mathrm{x}||}$$

When this result is substituted into the representation of *x* in [link], we obtain the formula
Equation:

$$\begin{aligned}\mathrm{x} &= ||\mathrm{x}||\ cos\ \theta_1\mathrm{e}_1 + ||\mathrm{x}||\ cos\ \theta_2\mathrm{e}_2 + \cdots + ||\mathrm{x}||\ cos\ \theta_n\mathrm{e}_n \\ &= ||\mathrm{x}||(\cos\,\theta_1\mathrm{e}_1 + \cos\,\theta_2\mathrm{e}_2 + \cdots + \cos\,\theta_n\mathrm{e}_n).\end{aligned}$$

This formula really shows that the vector x has "magnitude" $||\mathrm{x}||$ and direction $(\theta_1, \theta_2, ..., \theta_n)$ and that the magnitude and direction are sufficient to determine x. We call ($\cos\,\theta_1$, $\cos\,\theta_2$, ..., $\cos\,\theta_n$) the *direction cosines* for the vector x. In the three-dimensional case, they are illustrated in [link].

Exercise:

Problem:

Show that [link] agrees with the usual definition of an angle in two dimensions.

Exercise:

Problem: Find the cosine of the angle between x and y where

$$\text{a. x} = \begin{matrix}1\\0\\0\end{matrix}, \text{ y} = \begin{matrix}2\\2\\2\end{matrix};$$

$$\text{b. x} = \begin{matrix}1\\-1\\1\\-1\end{matrix}, \text{ y} = \begin{matrix}-1\\0\\1\\0\end{matrix};$$

$$\text{c. x} = \begin{matrix}2\\1\\-2\end{matrix}, \text{ y} = \begin{matrix}4\\2\\-4\end{matrix}$$

If we compare [link] and [link] we see that the direction vector u_x is composed of direction cosines:

Equation:

$$\text{u}_\text{x} = \cos\,\theta_1 \text{e}_1 + \cos\,\theta_2 \text{e}_2 + \cdots + \cos\,\theta_n \text{e}_n = \begin{matrix}\cos\theta_1\\\cos\theta_2\\\vdots\\\cos\theta_n\end{matrix}.$$

With this definition we can write [link] compactly as

Here x is written as the product of its magnitude $||\mathrm{x}||$ and its direction vector u_x. Now we can give an easy procedure to find a vector's direction angles:

(i) find $||\mathrm{x}||$;

(ii) calculate $\mathrm{u_x} = \dfrac{\mathrm{x}}{||\mathrm{x}||}$; and

(iii) take the arc cosines of the elements of u_x.

Step (iii) is often unnecessary; we are usually more interested in the direction vector (unit vector) u_x. Direction vectors are used in materials science in order to study the orientation of crystal lattices and in electromagnetic field theory to characterize the direction of propagation for radar and microwaves. You will find them of inestimable value in your courses on electromagnetic fields and antenna design.

Exercise:

Problem:

Sketch an arbitrary unit vector $\mathrm{u} \in \mathscr{R}^3$. Label the direction cosines and the components of u. S

Exercise:

Problem:

Compute the norm and the direction cosines for the vector $\mathrm{x} = \begin{matrix} 4 \\ 2 \\ 6 \end{matrix}$.

Exercise:

Problem:

Prove that the direction cosines for any vector satisfy the equality

Equation:

$$\cos^2 \theta_1 + \cos^2 \theta_2 + \cdots + \cos^2 \theta_n = 1.$$

What does this imply about the number of scalars needed to determine a vector $\mathrm{x} \in \mathscr{R}^n$?

Exercise:

Problem:

Astronomers use right ascension, declination, and distance to locate stars. On [link] these are, respectively, $-\varphi$, $\frac{\pi}{2} - \theta_3$, and $||\mathrm{x}||$. Represent $\mathrm{x} = (x_1, x_2, x_3)$ in terms of φ, θ_3, and $||\mathrm{x}||$ only. (That is, find equations that give φ, θ_3, and $||\mathrm{x}||$ in terms of x_1, x_2, and x_3, and find equations that give x_1, x_2, and x_3 in terms of φ, θ_3, and $||\mathrm{x}||$.)

Exercise:

Problem:

(MATLAB) Write a MATLAB function that accepts any vector $\mathrm{x} \in \mathscr{R}^n$ and computes its norm and direction cosines. Make x an input variable, and make $||\mathrm{x}||$ and u_x output variables.

Exercise:

Problem:

Let x and y denote two vectors in R^n with respective direction cosines ($\cos\ \theta_1, \cos\ \theta_2, ..., \cos\ \theta_n$) and ($\cos\ \varphi_1, \cos\ \varphi_2, ..., \cos\ \varphi_n$). Prove that ψ, the angle between x and y, is

Equation:

$$\cos\ \psi = \cos\ \theta_1\ \cos\ \varphi_1 + \cos\ \theta_2\ \cos\ \varphi_2 + \cdots + \cos\ \theta_n\ \cos\ \varphi_n.$$

Specialize this result to R^2 for insight.

Exercise:

Problem:

Compute the angle between the vectors $\mathrm{x} = \begin{matrix} 2 \\ 3 \\ 0 \end{matrix}$ and $\begin{matrix} 2 \\ 3 \\ 1 \end{matrix}$. Sketch the result.

Linear Algebra: Projections

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Orthogonality. When the angle between two vectors is $\pi/2\left(90^0\right)$, we say that the vectors are *orthogonal*. A quick look at the definition of angle (Equation 12 from "Linear Algebra: Direction Cosines") leads to this equivalent definition for orthogonality:
Equation:

$$(\mathrm{x},\mathrm{y})=0 \Leftrightarrow \mathrm{x} \text{ and } y \text{ are orthogonal.}$$

For example, in Figure 1(a), the vectors $\mathrm{x}=\begin{bmatrix}3\\1\end{bmatrix}$ and $\mathrm{y}=\begin{bmatrix}-2\\6\end{bmatrix}$ are clearly orthogonal, and their inner product is zero:
Equation:

$$(\mathrm{x},\mathrm{y})=3(-2)+1(6)=0.$$

In Figure 1(b), the vectors $\mathrm{x}=\begin{matrix}3\\1\\0\end{matrix}$, $\mathrm{y}=\begin{matrix}-2\\6\\0\end{matrix}$, and $\mathrm{z}=\begin{matrix}0\\0\\4\end{matrix}$ are mutually orthogonal, and the inner product between each pair is zero:
Equation:

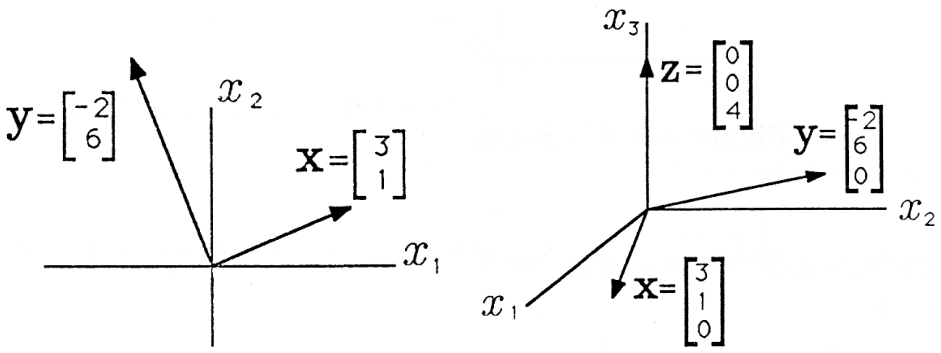
$$(\mathrm{x},\mathrm{y})=3(-2)+1(6)+0(0)=0$$

Equation:

$$(\mathrm{x}, \mathrm{z}) = 3(0) + 1(0) + 0(4) = 0$$

Equation:

$$(\mathrm{y}, \mathrm{z}) = -2(0) + 6(0) + 0(4) = 0.$$



Orthogonality of Vectors

Exercise:

Problem: Which of the following pairs of vectors is orthogonal:

$$\text{a. } \mathrm{x} = \begin{matrix} 1 \\ 0 \\ 1 \end{matrix}, \ \mathrm{y} = \begin{matrix} 0 \\ 1 \\ 0 \end{matrix};$$

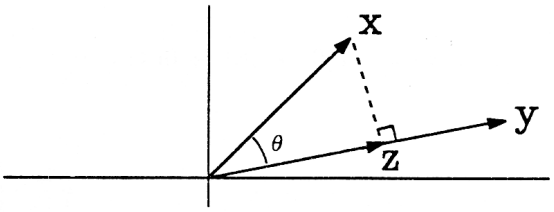
$$\text{b. } \mathrm{x} = \begin{matrix} 1 \\ 1 \\ 0 \end{matrix}, \ \mathrm{y} = \begin{matrix} 1 \\ 1 \\ 1 \end{matrix};$$

$$\text{c. } \mathrm{x} = \mathrm{e}_1, \mathrm{y} = \mathrm{e}_3;$$

$$\text{d. } \mathrm{x} = \begin{bmatrix} a \\ b \end{bmatrix}, \mathrm{y} = \begin{bmatrix} -b \\ a \end{bmatrix}?$$

We can use the inner product to find the projection of one vector onto another as illustrated in Figure 2. Geometrically we find the projection of x

onto y by dropping a perpendicular from the head of x onto the line containing y. The perpendicular is the dashed line in the figure. The point where the perpendicular intersects y (or an extension of y) is the projection of x onto y, or the component of x along y. Let's call it z.



Component of One Vector along Another

Exercise:

Problem:

Draw a figure like Figure 2 showing the projection of y onto x.

The vector z lies along y, so we may write it as the product of its norm $||z||$ and its direction vector u_y :

Equation:

$$\mathrm{z} = ||\mathrm{z}||\mathrm{u_y} = ||\mathrm{z}||\frac{\mathrm{y}}{||\mathrm{y}||}.$$

But what is norm $||z||$? From Figure 2 we see that the vector x is just z, plus a vector v that is orthogonal to y:

Equation:

$$\mathrm{x} = \mathrm{z} + \mathrm{v}, \ \ (\mathrm{v}, \mathrm{y}) = 0.$$

Therefore we may write the inner product between x and y as

Equation:

$$(\mathrm{x},\mathrm{y}) = (\mathrm{z}+\mathrm{v},\mathrm{y}) = (\mathrm{z},\mathrm{y}) + (\mathrm{v},\mathrm{y}) = (\mathrm{z},\mathrm{y}).$$

But because z and y both lie along y, we may write the inner product (x,y) as

Equation:

$$\begin{aligned}(\mathrm{x},\mathrm{y}) &= (\mathrm{z},\mathrm{y}) = (||\mathrm{z}||\mathrm{u}_\mathrm{y}, ||\mathrm{y}||\mathrm{u}_\mathrm{y}) = ||\mathrm{z}||||\mathrm{y}||\,(\mathrm{u}_\mathrm{y},\mathrm{u}_\mathrm{y}) \\ &= ||\mathrm{z}||||\mathrm{y}||||\mathrm{u}_\mathrm{y}||^2 = ||\mathrm{z}||||\mathrm{y}||.\end{aligned}$$

From this equation we may solve for $||\mathrm{z}|| = \dfrac{(\mathrm{x},\mathrm{y})}{||\mathrm{y}||}$ and substitute $||\mathrm{z}||$ into Equation 6 to write z as

Equation:

$$\begin{aligned}\mathrm{z} &= ||\mathrm{z}||\tfrac{\mathrm{y}}{||\mathrm{y}||} \\ &= \tfrac{(\mathrm{x},\mathrm{y})}{||\mathrm{y}||}\tfrac{\mathrm{y}}{||\mathrm{y}||} = \tfrac{(\mathrm{x},\mathrm{y})}{(\mathrm{y},\mathrm{y})}\mathrm{y}.\end{aligned}$$

Equation 10 is what we wanted–an expression for the projection of x onto y in terms of x and y.

Exercise:

Problem:

Show that $||\mathrm{z}||$ and z may be written in terms of cos θ for θ as illustrated in Figure 2:

Equation:

$$||\mathrm{z}|| = ||\mathrm{x}||\ cos\ \theta$$

Equation:

$$\mathrm{z} = \frac{||\mathrm{x}||\ \cos\theta}{||\mathrm{y}||}\mathrm{y}.$$

Orthogonal Decomposition. You already know how to decompose a vector in terms of the unit coordinate vectors,
Equation:

$$\mathrm{x} = (\mathrm{x}, \mathrm{e}_1)\mathrm{e}_1 + (\mathrm{x}, \mathrm{e}_2)\mathrm{e}_2 + \cdots + (\mathrm{x}, \mathrm{e}_n)\mathrm{e}_n.$$

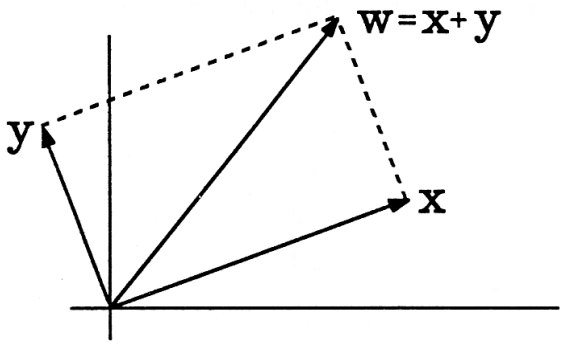
In this equation, $(\mathrm{x}, \mathrm{e}_k)\mathrm{e}_k$ is the component of x along e_k, or the projection of x onto e_k, but the set of unit coordinate vectors is not the only possible basis for decomposing a vector. Let's consider an arbitrary pair of orthogonal vectors x and y:
Equation:

$$(\mathrm{x}, \mathrm{y}) = 0.$$

The sum of x and y produces a new vector w, illustrated in Figure 3, where we have used a two-dimensional drawing to represent n dimensions. The norm squared of w is

$$\begin{aligned} ||\mathrm{w}||^2 &= (\mathrm{w}, \mathrm{w}) = [(\mathrm{x} + \mathrm{y}), (\mathrm{x} + \mathrm{y})] = (\mathrm{x}, \mathrm{x}) + (\mathrm{x}, \mathrm{y}) + (\mathrm{y}, \mathrm{x}) + (\mathrm{y}, \mathrm{y}) \\ &= ||\mathrm{x}||^2 + ||\mathrm{y}||^2. \end{aligned}$$

This is the Pythagorean theorem in n dimensions! The length squared of w is just the sum of the squares of the lengths of its two orthogonal components.



Sum of Orthogonal Vectors

The projection of w onto x is x, and the projection of w onto y is y:
Equation:

$$\mathrm{w} = (1)\mathrm{x} + (1)\mathrm{y}.$$

If we scale w by a to produce the vector $\mathrm{z} = a\mathrm{w}$, the orthogonal decomposition of z is
Equation:

$$\mathrm{z} = a\mathrm{w} = (\mathrm{a})\mathrm{x} + (\mathrm{a})\mathrm{y}.$$

Let's turn this argument around. Instead of building w from orthogonal vectors x and y, let's begin with arbitrary w and x and see whether we can compute an orthogonal decomposition. The projection of w onto x is found from Equation 10:
Equation:

$$\mathrm{w}_\mathrm{x} = \frac{(\mathrm{w}, \mathrm{x})}{(\mathrm{x}, \mathrm{x})}\mathrm{x}.$$

But there must be another component of w such that w is equal to the sum of the components. Let's call the unknown component w_y. Then
Equation:

$$\mathrm{w} = \mathrm{w}_\mathrm{x} + \mathrm{w}_\mathrm{y}.$$

Now, since we know w and w_x already, we find w_y to be
Equation:

$$\mathrm{w}_\mathrm{y} = \mathrm{w} - \mathrm{w}_\mathrm{x} = \mathrm{w} - \frac{(\mathrm{w}, \mathrm{x})}{(\mathrm{x}, \mathrm{x})}\mathrm{x}.$$

Interestingly, the way we have decomposed w will always produce w_x and $w_{>\mathrm{y}}$ orthogonal to each other. Let's check this:
Equation:

$$\begin{aligned}
(\mathrm{w_x}, \mathrm{w_y}) &= \tfrac{(\mathrm{w,x})}{(\mathrm{x,x})}\mathrm{x},\ \ \mathrm{w} - \tfrac{(\mathrm{w,x})}{(\mathrm{x,x})}\mathrm{x} \\
&= \tfrac{(\mathrm{w,x})}{(\mathrm{x,x})}(\mathrm{x}, \mathrm{w}) - \tfrac{(\mathrm{w,x})^2}{(\mathrm{x,x})^2}(\mathrm{x}, \mathrm{x}) \\
&= \tfrac{(\mathrm{w,x})^2}{(\mathrm{x,x})} - \tfrac{(\mathrm{w,x})^2}{(\mathrm{x,x})} \\
&= 0.
\end{aligned}$$

To summarize, we have taken two arbitrary vectors, w and x, and decomposed w into a component in the direction of x and a component orthogonal to x.

Linear Algebra: Other Norms

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Sometimes we find it useful to use a different definition of distance, corresponding to an alternate norm for vectors. For example, consider the *l-norm* defined as

Equation:

$$||\mathrm{x}||_1 = (|x_1|+|x_2|+\cdots+|x_n|),$$

where $|x_i|$ is the magnitude of component x_i. There is also the *sup-norm*, the "supremum" or maximum of the components $x_1, ..., x_n$:

Equation:

$$||\mathrm{x}||_{\mathrm{sup}} = \max(|x_1|, |x_2|, ..., |x_n|).$$

The following examples illustrate what the Euclidean norm, the l-norm, and the sup-norm look like for typical vectors.

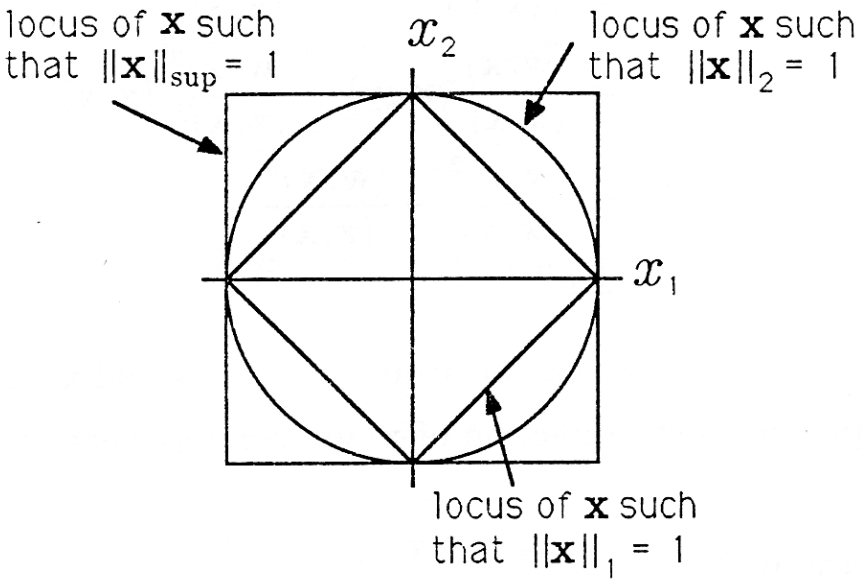
Example:

Consider the vector $\mathrm{x} = \begin{matrix} -3 \\ 1 \\ 2 \end{matrix}$. Then

i. $||\mathrm{x}|| = \left[(-3)^2 + (1)^2 + (2)^2\right]^{1/2} = (14)^{1/2}$;
ii. $||\mathrm{x}||_1 = (|-3| + |1| + |2|) = 6$; and
iii. $||\mathrm{x}||_{\text{sup}} = \max(|-3|, |1|, |2|) = 3$.

Example:

Figure 1 shows the locus of two-component vectors $\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ with the property that $||\mathrm{x}|| = 1$, $||\mathrm{x}||_1 = 1$, or $||\mathrm{x}||_{\text{sup}} = 1$.



Locus of Two-Dimensional Vectors Whose Various Norms Are 1

The next example shows how the l-norm is an important part of city life.

Example:
The city of Metroville was laid out by mathematicians as shown in Figure 2. A person at the intersection of Avenue 0 and Street -2 (point A) is clearly two blocks from the center of town (point C). This is consistent with both the Euclidean norm
Equation:

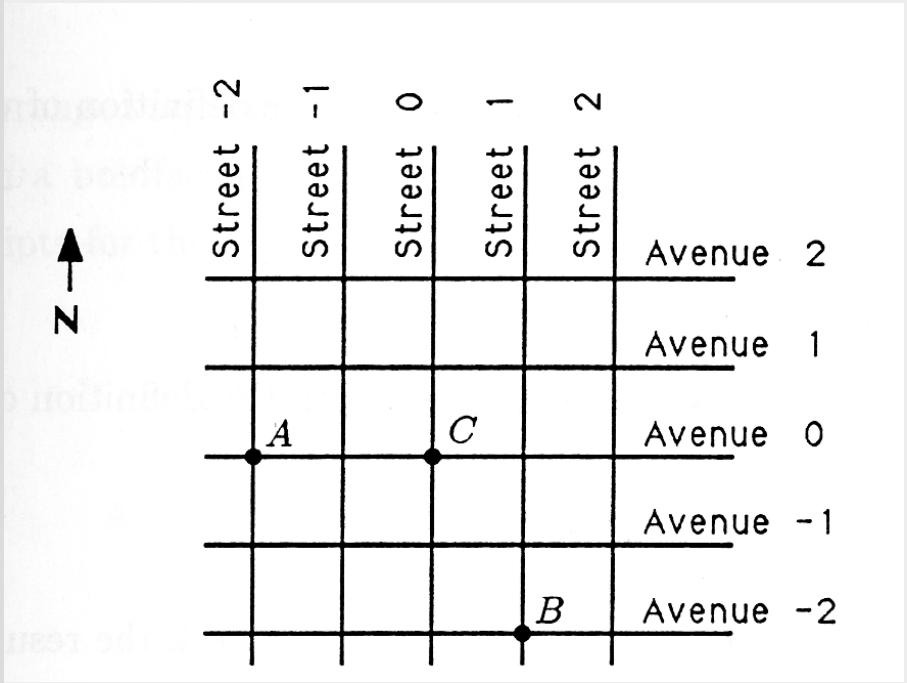
$$||A|| = \sqrt{0^2 + (-2)^2} = \sqrt{4} = 2$$

and the l-norm
Equation:

$$||A||_1 = (|0| + |-2|) = 2.$$

But how far from the center of town is point B at the intersection of Avenue-2 and Street 1? According to the Euclidean norm, the distance is
Equation:

$$||B|| = \sqrt{(-2)^2 + (1)^2} = \sqrt{5}.$$



Metroville, U.S.A.

While it is true that point B is $\sqrt{5}$ blocks from C, it is also clear that the trip would be three blocks by any of the three shortest routes on roads. The appropriate norm is the l-norm:

Equation:

$$\left|1^{B}\right|\left|_{1}=(|-2|+|1|)=3.\right.$$

Even more generally, we can define a norm for each value of p from 1 to infinity. The so-called *p-norm* is

Equation:

$$|\mathrm{Ix}|\left|_{p}=\left(|x_{1}\right|^{p}+|x_{2}\right|^{p}+\cdots+|x_{n}|^{p})^{1/p}.\right.$$

Exercise:

Problem:

Show that the Euclidean norm is the same as the p-norm with $p = 2$ and that the 1-norm is the p-norm with $p = 1$. (It can also be shown that the sup-norm is like a p-norm with $p = \infty$.)

DEMO 4.1 (MATLAB). From the command level of MATLAB, type the following lines:

```
>> x = [1;3;-2;4]
>> y = [0;1;2;-0.5]
>> x - y
```

Check to see whether the answer agrees with the definition of vector subtraction. Now type

```
>> a = -1.5
>> a * x
```

Check the answer to see whether it agrees with the definition of scalar multiplication. Now type

```
>> x' * y
```

This is how MATLAB does the inner product. Check the result. Type

```
>> norm(y)
>> sqrt(y' * y)

>> norm(y,1)
>> norm(y' * y)
```

Now type your own MATLAB expression to find the cosine of the angle between vectors x and y. Put the result in variable t. Then find the angle θ by typing

```
>> theta = acos(t)
```

The angle θ is in radians. You may convert it to degrees if you wish by multiplying it by $180/\pi$:

```
>> theta = theta * (180/pi)
```

Linear Algebra: Matrices

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

The word *matrix* dates at least to the thirteenth century, when it was used to describe the rectangular copper tray, or matrix, that held individual leaden letters that were packed into the matrix to form a page of composed text. Each letter in the matrix, call it a_{ij}, occupied a unique position (ij) in the matrix. In modern day mathematical parlance, a matrix is a collection of numbers arranged in a two-dimensional array (a rectangle). We indicate a matrix with a boldfaced capital letter and the constituent elements with double subscripts for the row and column:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix} \leftarrow \text{second row}$$

$$\uparrow$$

third column

Matrix

In this equation A is an m x n matrix, meaning that A has m horizontal rows and n vertical columns. As an extension of the previously used notation, we write $A \in \mathscr{R}^{mxn}$ to show that A is a matrix of size m x n with $a_{ij} \in \mathscr{R}$. The scalar element a_{ij} is located in the matrix at the ith row and the jth column. For example, a_{23} is located in the second row and the third column as illustrated in [link].

The *main diagonal* of any matrix consists of the elements a_{ii}. (The two subscripts are equal.) The main diagonal runs southeast from the top left corner of the matrix, but it does not end in the lower right corner unless the matrix is square $(\in \mathscr{R}^{m \times m})$.

The *transpose* of a matrix $A \in \mathscr{R}^{m \times n}$ is another matrix B whose element in row j and column i is $b_{ji} = a_{ij}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$. We write $B = A^T$ to indicate that B is the transpose of A. In MATLAB, transpose is denoted by `A'`. A more intuitive way of describing the transpose operation is to say that it flips the matrix about its main diagonal so that rows become columns and columns become rows.

Exercise:

Problem: If $A \in \mathscr{R}^{m \times n}$, then $A^T \in \underline{?}$. Find the transpose of the matrix $A = \begin{matrix} 2 & 1 \\ 5 & 4 \\ 7 & 9 \end{matrix}$.

Matrix Addition and Scalar Multiplication. Two matrices of the same size (in both dimensions) may be added or subtracted in the same way as vectors, by adding or subtracting the corresponding elements. The equation $C = A \pm B$ means that for each *i* and *j*, $c_{ij} = a_{ij} \pm b_{ij}$. Scalar multiplication of a matrix multiplies each element of the matrix by the scalar:

Equation:

$$\mathrm{aX} = \begin{matrix} ax_{11} & ax_{12} & ax_{13} & \ldots & ax_{1n} \\ ax_{21} & ax_{22} & ax_{23} & \ldots & ax_{2n} \\ ax_{31} & ax_{32} & ax_{33} & \ldots & ax_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ ax_{m1} & ax_{m2} & ax_{m3} & \ldots & ax_{mn} \end{matrix} .$$

Matrix Multiplication. A vector can be considered a matrix with only one column. Thus we intentionally blur the distinction between $\mathscr{R}^{n\times 1}$ and R^n. Also a matrix can be viewed as a collection of vectors, each column of the matrix being a vector:
Equation:

$$\mathrm{A} = \begin{matrix} | & | & & | \\ \mathrm{a}_1 & \mathrm{a}_2 & \ldots & \mathrm{a}_n \\ | & | & & | \end{matrix} = \begin{matrix} a_{11} & a_{12} & a_{1n} \\ a_{21} & a_{22} & a_{2n} \\ | & | & | \\ a_{m1} & a_{m2} & a_{mn} \end{matrix} .$$

In the transpose operation, columns become rows and vice versa. The transpose of an $n \times 1$ matrix, a *column vector*, is a $1 \times n$ matrix, a *row vector*:
Equation:

$$\mathrm{x} = \begin{matrix} x_1 \\ x_2 \\ | \\ x_n \end{matrix} ; \mathrm{x}^T = [x_1 x_2 x_n].$$

Now we can define matrix-matrix multiplication in terms of inner products of vectors. Let's begin with matrices $\mathrm{A} \in \mathscr{R}^{m\times n}$ and $\mathrm{B} \in \mathscr{R}^{n\times p}$. To find the product AB, first divide each matrix into column vectors and row vectors as follows:

$$\mathbf{A} = \begin{bmatrix} \begin{bmatrix} | \\ \mathbf{a}_1 \\ | \end{bmatrix} & \begin{bmatrix} | \\ \mathbf{a}_2 \\ | \end{bmatrix} & \cdots & \begin{bmatrix} | \\ \mathbf{a}_n \\ | \end{bmatrix} \end{bmatrix} = \begin{bmatrix} [- & \boldsymbol{\alpha}_1^T & -] \\ [- & \boldsymbol{\alpha}_2^T & -] \\ & \vdots & \\ [- & \boldsymbol{\alpha}_m^T & -] \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \begin{bmatrix} | \\ \mathbf{b}_1 \\ | \end{bmatrix} & \begin{bmatrix} | \\ \mathbf{b}_2 \\ | \end{bmatrix} & \cdots & \begin{bmatrix} | \\ \mathbf{b}_p \\ | \end{bmatrix} \end{bmatrix} = \begin{bmatrix} [- & \boldsymbol{\beta}_1^T & -] \\ [- & \boldsymbol{\beta}_2^T & -] \\ & \vdots & \\ [- & \boldsymbol{\beta}_n^T & -] \end{bmatrix} .$$

Thus a_i is the ith column of *A* and α_j^T is the jth row of A. For matrix multiplication to be defined, the width of the first matrix must match the length of the second one so that all rows α_i^T and columns b_i have the same number of elements n. The matrix product, $\mathrm{C} = \mathrm{AB}$, is an $m \times p$ matrix defined by its elements as $\mathrm{c}_{ij} = (\alpha_i, \mathrm{b}_j)$. In words, each element of the product matrix, c_{ij}, is the inner product of the ith row of the first matrix and the jth column of the second matrix.

For n-vectors x and y, the matrix product $\mathrm{x}^T\mathrm{y}$ takes on a special significance. The product is, of course, a 1×1 matrix (a scalar). The special significance is that $\mathrm{x}^T\mathrm{y}$ is the inner product of x and y:

$$\begin{bmatrix} — & \mathbf{x}^T & — \end{bmatrix} \begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix} = (\mathbf{x}, \mathbf{y}).$$

Thus the notation $\mathrm{x}^T\mathrm{y}$ is often used in place of (x, y). Recall from Demo 1 from "Linear Algebra: Other Norms" that MATLAB uses `x'*y` to denote inner product.

Another special case of matrix multiplication is the *outer product*. Like the inner product, it involves two vectors, but this time the result is a matrix:

$$\mathbf{A} = \begin{bmatrix} | \\ \mathbf{x} \\ | \end{bmatrix} \begin{bmatrix} — & \mathbf{y}^T & — \end{bmatrix} = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & & \vdots \\ x_my_1 & x_my_2 & \cdots & x_my_n \end{bmatrix}.$$

In the outer product, the inner products that define its elements are between one-dimensional row vectors of x and one-dimensional column vectors of y^T, meaning the (i, j) element of A is x_iy_j.

Exercise:

Problem: Find $\mathrm{C} = \mathrm{AB}$ where A and B are given by

a. $\mathrm{A} = \begin{bmatrix} 1 & -1 & 2 \\ 3 & 0 & 5 \end{bmatrix}$, $\mathrm{B} = \begin{matrix} 0 & -2 & 1 & -3 \\ 4 & 2 & 2 & 0 \\ 2 & -2 & 3 & 1 \end{matrix}$;

b. $\mathrm{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\mathrm{B} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$;

c. $\mathrm{A} = \begin{matrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{matrix}$, $\mathrm{B} = \begin{matrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{matrix}$.

There are several other equivalent ways to define matrix multiplication, and a careful study of the following discussion should improve your understanding of matrix multiplication. Consider $\mathrm{A} \in \mathscr{R}^{m \times n}$, $\mathrm{B} \in \mathscr{R}^{n \times p}$, and $\mathrm{C} = \mathrm{AB}$ so that $\mathrm{C} \in \mathscr{R}^{m \times p}$. In pictures, we have

Equation:

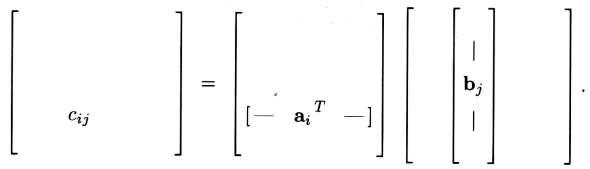
$$m \begin{bmatrix} p \\ \mathrm{C} \end{bmatrix} = m \begin{bmatrix} n \\ \mathrm{A} \end{bmatrix} \begin{bmatrix} p \\ \mathrm{B} \end{bmatrix} n.$$

In our definition, we represent C on an entry-by-entry basis as

Equation:

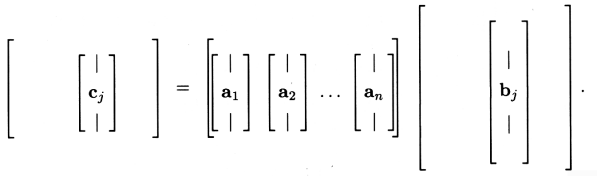
$$\mathrm{c}_{ij} = (\alpha_i, \mathrm{b}_j) = \sum_{k=1}^{n} \mathrm{a}_{ik} b_{kj}.$$

In pictures,



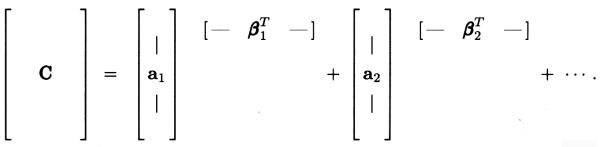
You will prove in Exercise 3 that we can also represent C on a column basis:
Equation:

$$\mathrm{c}_j = \sum_{k=1}^{n} \mathrm{a}_k b_{kj}.$$



Finally, C can be represented as a sum of matrices, each matrix being an outer product:
Equation:

$$\mathrm{C} = \sum_{i=1}^{n} \mathrm{a}_i \beta_i^T$$



A numerical example should help clarify these three methods.

Example:
Let
Equation:

$$A = \begin{matrix} 1 & 2 & 1 & 3 \\ 2 & 1 & 2 & 4 \\ 3 & 3 & 2 & 1 \end{matrix}, \quad B = \begin{matrix} 1 & 2 & 1 \\ 2 & 2 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 1 \end{matrix}.$$

Using the first method of matrix multiplication, on an entry-by-entry basis, we have
Equation:

$$\mathrm{c}_{ij} = \sum_{k=1}^{4} \mathrm{a}_{ik} b_{kj}$$

or
Equation:

$$\mathrm{C} = \begin{matrix} (1\cdot 1+2\cdot 2+1\cdot 1+3\cdot 2) & (1\cdot 2+2\cdot 2+1\cdot 3+3\cdot 1) & (1\cdot 1+2\cdot 1+1\cdot 2+3\cdot 1) \\ (2\cdot 1+1\cdot 2+2\cdot 1+4\cdot 2) & (2\cdot 2+1\cdot 2+2\cdot 3+4\cdot 1) & (2\cdot 1+1\cdot 1+2\cdot 2+4.1) \\ (3\cdot 1+3\cdot 2+2\cdot 1+1\cdot 2) & (3\cdot 2+3\cdot 2+2\cdot 3+1\cdot 1) & (3\cdot 1+3\cdot 1+2\cdot 2+1\cdot 1) \end{matrix}$$

or

Equation:

$$\mathrm{C} = \begin{matrix} 12 & 12 & 8 \\ 14 & 16 & 11 \\ 13 & \mathrm{l}9 & \mathrm{l}1 \end{matrix}.$$

On a column basis,

Equation:

$$\mathrm{c}_j = \sum_{k=1}^{4} \mathrm{a}_k b_{kj}$$

$$\mathrm{c}_1 = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} 1 + \begin{matrix} 2 \\ 1 \\ 3 \end{matrix} 2 + \begin{matrix} 1 \\ 2 \\ 2 \end{matrix} 1 + \begin{matrix} 3 \\ 4 \\ 1 \end{matrix} 2; \mathrm{c}_2 = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} 2 + \begin{matrix} 2 \\ 1 \\ 3 \end{matrix} 2 + \begin{matrix} 1 \\ 2 \\ 2 \end{matrix} 3 + \begin{matrix} 3 \\ 4 \\ 1 \end{matrix} 1; \mathrm{c}_3 = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} 1 + \begin{matrix} 2 \\ 1 \\ 3 \end{matrix} 1 + \begin{matrix} 1 \\ 2 \\ 2 \end{matrix} 2 +$$

Collecting terms together, we have

Equation:

$$\mathrm{C} = [\mathrm{c}_1 \mathrm{c}_2 \mathrm{c}_3]$$

$$\mathrm{C} = \begin{matrix} (1\cdot 1+2\cdot 2+1\cdot 1+3\cdot 2) & (1\cdot 2+2\cdot 2+1\cdot 3+3\cdot 1) & (1\cdot 1+2\cdot 1+1\cdot 2+3\cdot 1) \\ (2\cdot 1+1\cdot 2+2\cdot 1+4\cdot 2) & (2\cdot 2+1\cdot 2+2\cdot 3+4\cdot 1) & (2\cdot 1+1\cdot 1+2\cdot 2+4.1) \\ (3\cdot 1+3\cdot 2+2\cdot 1+1\cdot 2) & (3\cdot 2+3\cdot 2+2\cdot 3+1\cdot 1) & (3\cdot 1+3\cdot 1+2\cdot 2+1\cdot 1) \end{matrix}$$

On a matrix-by-matrix basis,

Equation:

$$\mathrm{C} = \sum_{i=1}^{4} \mathrm{a}_i \beta_i^T$$

Equation:

$$C = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{matrix} 1 & 2 & 1 \end{matrix} + \begin{matrix} 2 \\ 1 \\ 3 \end{matrix} \begin{matrix} 2 & 2 & 1 \end{matrix} + \begin{matrix} 1 \\ 2 \\ 2 \end{matrix} \begin{matrix} 1 & 3 & 2 \end{matrix} + \begin{matrix} 3 \\ 4 \\ 1 \end{matrix} \begin{matrix} 1 & 1 & 1 \end{matrix}$$

Equation:

$$= \begin{matrix} 1\cdot 1 & 1\cdot 2 & 1\cdot 1 \\ 2\cdot 1 & 2\cdot 2 & 2\cdot 1 \\ 3\cdot 1 & 3\cdot 2 & 3\cdot 1 \end{matrix} + \begin{matrix} 2\cdot 2 & 2\cdot 2 & 2\cdot 1 \\ 1\cdot 2 & 1\cdot 2 & 1\cdot 1 \\ 3\cdot 2 & 3\cdot 2 & 3\cdot 1 \end{matrix} + \begin{matrix} 1\cdot 1 & 1\cdot 3 & 1\cdot 2 \\ 2\cdot 1 & 2\cdot 3 & 2\cdot 2 \\ 2\cdot 1 & 2\cdot 3 & 2\cdot 2 \end{matrix} + \begin{matrix} 3\cdot 2 & 3\cdot 1 & 3\cdot 1 \\ 4\cdot 2 & 4\cdot 1 & 4\cdot 1 \\ 1\cdot 2 & 1\cdot 1 & 1\cdot 1 \end{matrix}$$

Equation:

$$\mathrm{C} = \begin{matrix} (1\cdot 1+2\cdot 2+1\cdot 1+3\cdot 2) & (1\cdot 2+2\cdot 2+1\cdot 3+3\cdot 1) & (1\cdot 1+2\cdot 1+1\cdot 2+3\cdot 1) \\ (2\cdot 1+1\cdot 2+2\cdot 1+4\cdot 2) & (2\cdot 2+1\cdot 2+2\cdot 3+4\cdot 1) & (2\cdot 1+1\cdot 1+2\cdot 2+4.1) \\ (3\cdot 1+3\cdot 2+2\cdot 1+1\cdot 2) & (3\cdot 2+3\cdot 2+2\cdot 3+1\cdot 1) & (3\cdot 1+3\cdot 1+2\cdot 2+1\cdot 1) \end{matrix}$$

as we had in each of the other cases. Thus we see that the methods are equivalent-simply different ways of organizing the same computation!

Exercise:

Problem:

Prove that Equations 9, 11, and 13 are equivalent definitions of matrix multiplication. That is, if $\mathrm{C} = \mathrm{AB}$ where $\mathrm{A} \in \mathscr{R}^{m \times n}$ and $\mathrm{B} \in \mathscr{R}^{n \times p}$, show that the matrix-matrix product can also be defined by
Equation:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj},$$

and, if c_k is the kth column of C and a_k is the kth column of A, then
Equation:

$$\mathrm{c}_j = \sum_{k=1}^{n} \mathrm{a}_k b_{kj}.$$

Show that the matrix C may also be written as the , "sum of outer products"
Equation:

$$\mathrm{C} = \sum_{k=1}^{n} \mathrm{a}_k \beta_k^T.$$

Write out the elements in a typical outer product $\mathrm{a}_k \beta_k^T$.

Exercise:

Problem:

Given $A \in \mathscr{R}^{m \times n}$, $\mathrm{B} \in \mathscr{R}^{p \times q}$, and $\mathrm{C} \in \mathscr{R}^{r \times s}$, for each of the following postulates, either prove that it is true or give a counterexample showing that it is false:

a. $\left(\mathrm{A}^T\right)^T = \mathrm{A}$.
b. $AB = \mathrm{BA}$ when $n = p$ and $m = q$. Is matrix multiplication commutative?
c. $\mathrm{A}(\mathrm{B} + \mathrm{C}) = AB + \mathrm{AC}$ when $n = p = r$ and $q = s$. Is matrix multiplication distributive over addition?
d. $(\mathrm{AB}\,)^T = \mathrm{B}^T \mathrm{A}^T$ when $n = p$.
e. $(AB)C = A(BC)$ when $n = p$ and $q = r$. Is matrix multiplication associative?

Example:
Rotation
We know from the chapter on complex numbers that a complex number $z_1 = x_1 + jy_1$ may be rotated by angle θ in the complex plane by forming the product
Equation:

$$z_2 = e^{j\theta} z_1.$$

When written out, the real and imaginary parts of z_2 are
Equation:

$$\begin{aligned} z_2 &= (\cos\theta + j\sin\theta)(x_1 + jy_1) \\ &= (\cos\theta)x_1 - (\sin\theta)y_1 + j[(\sin\theta)x_1 + (\cos\theta)y_1]. \end{aligned}$$

If the real and imaginary parts of z_1 and z_2 are organized into vectors z_1 and z_2 as in the chapter on complex numbers, then rotation may be carried out with the matrix-vector multiply
Equation:

$$\mathrm{z}_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}.$$

We call the matrix $\mathrm{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ a *rotation matrix*.

Exercise:

Problem: Let $\mathrm{R}(\theta)$ denote a 2×2 rotation matrix. Prove and interpret the following two properties:

a. $\mathrm{R}^T(\theta) = \mathrm{R}(-\theta)$;
b. $\mathrm{R}^T(\theta)\mathrm{R}(\theta) = \mathrm{R}(\theta)\mathrm{R}^T(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

Linear Algebra: Solving Linear Systems of Equations

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

We are now equipped to set up systems of linear equations as matrix- vector equations so that they can be solved in a standard way on a computer. Suppose we want to solve the equations from Example 1 from "Linear Algebra: Introduction" for x_1 and x_2 using a computer. Recall that Equations 1 and 2 from Linear Algebra: Introduction are
Equation:

$$\begin{aligned} x_1 + x_2 &= 85 \\ \frac{x_1}{1.2} &= \frac{x_2}{1.5-1.2} \end{aligned}$$

The first step is to arrange each equation with all references to x_1 in the first column, all references to x_2 in the second column, etc., and all constants on the right-hand side:
Equation:

$$\begin{aligned} x_1 + x_2 &= 85 \\ 0.3x_1 - 1.2x_2 &= \ \ 0. \end{aligned}$$

Then the equations can be converted to a single matrix-vector equation. The coefficients form a matrix, keeping their same relative positions, and the variables and constants each form a vector:
Equation:

$$\begin{bmatrix} 1 & 1 \\ 0.3 & -1.2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 85 \\ 0 \end{bmatrix}.$$

Exercise:

Problem:

Verify by the rules of matrix multiplication that the system of equations in 2 in equivalent to the matrix equation in 3.

Equation 3 is of the general form
Equation:

$$Ax = \mathrm{b}$$

where in this case
Equation:

$$\mathrm{A} = \begin{bmatrix} 1 & 1 \\ 0.3 & -1.2 \end{bmatrix}, \mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \ \mathrm{b} = \begin{bmatrix} 85 \\ 0 \end{bmatrix}.$$

Given any $A \in \mathscr{R}^{n \times n}$ and $\mathrm{b} \in \mathscr{R}^n$, MATLAB can solve Equation 4 for x (as long as a solution exists). Key ideas in the solution process are the *identity* matrix and the *inverse* of a matrix.

When the matrix A is the 1×1 matrix a, the vector x is the l-vector x, and the vector b is the l-vector b, then the matrix equation $Ax = \mathrm{b}$ becomes the scalar equation
Equation:

$$ax = b.$$

The scalar a^{-1} is the inverse of the scalar a, so we may multiply on both sides of Equation 6 to obtain the result
Equation:

$$a^{-1}(ax) = a^{-1}b$$
$$1x = a^{-1}b.$$

We would like to generalize this simple idea to the matrix equation $Ax = \mathrm{b}$ so that we can find an inverse of the matrix A, called A^{-1}, and write
Equation:

$$\mathrm{A}^{-1}\big(Ax\big) = \mathrm{A}^{-1}\mathrm{b}$$
$$Ix = \mathrm{A}^{-1}\mathrm{b}.$$

In this equation the matrix I is the identity matrix
Equation:

$$\mathrm{I} = \begin{matrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{matrix}.$$

It is clear that the identity matrix I and the inverse of a matrix, A^{-1}, play a key role in the solution of linear equations. Let's study them in more detail.

The Matrix Identity. When we multiply a scalar by 1, we get back that same scalar. For this reason, the number 1 is called the *multiplicativeidentity* element. This may seem trivial, but the generalization to matrices is more interesting. The question is, is there a matrix that, when it multiplies another matrix, does not change the other matrix? The answer is yes. The matrix is called the *identity matrix* and is denoted by I. The identity matrix is always square, with whatever size is appropriate for the matrix multiplication. The identity matrix has l's on its main diagonal and $0'$s everywhere else. For example,
Equation:

$$I_5 = \begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{matrix}.$$

The subscript 5 indicates the size. In terms of the unit coordinate vectors e_i, we can also write the $n \times n$ identity matrix as

Equation:

$$\mathrm{I}_n = \begin{matrix} | & | & & | \\ \mathrm{e}_1 & \mathrm{e}_2 & \ldots & \mathrm{e}_n \\ | & | & & | \end{matrix}.$$

For any matrix $\mathrm{A} \in \mathscr{R}^{n \times p}$, we have

Equation:

$$\mathrm{A} = \mathrm{I}_n \mathrm{A}.$$

For $p = 1$, we obtain the following special form for any vector $\mathrm{x} \in \mathscr{R}^n$:

Equation:

$$\mathrm{x} = \mathrm{I}_n \mathrm{x}.$$

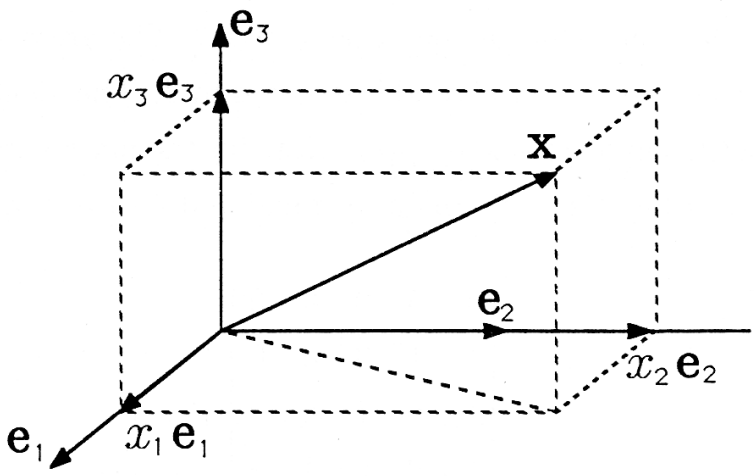
This last equation can be written as the sum

Equation:

$$\mathrm{x} = \sum_{i=1}^{n} x_i \mathrm{e}_i.$$

This is a special case of one of the results you proved in Exercise 3 from "Linear Algebra: Matrices". Figure 1 illustrates Equation 14 and shows how

the vector x can be broken into component vectors $x_i \mathrm{e}_i$, each of which lies in the direction of one coordinate axis. The values of the x_i's are the coordinates of x, and their magnitudes are also the lengths of the component vectors.



Breaking a Vector into Components

Exercise:

Problem:

Use Equation 13 and the rules for matrix multiplication to show that $\mathrm{x} \in \mathscr{R}^n$ may also be written as

Equation:

$$\mathrm{x} = \sum_{i=1}^{n} (\mathrm{x}, \mathrm{e}_i)\mathrm{e}_i.$$

This verifies Equation 11 from "Linear Algebra: Direction Cosines".

When the product of two numbers is 1 (the identity element), we say that they are inverses of each other, like 2 and 0.5. Likewise, we say that two *square* matrices are *inverses* of each other if their product is the identity matrix:

Equation:

$$AB = \mathrm{I} \Leftrightarrow \mathrm{B} = \mathrm{A}^{-1}.$$

An interesting and useful result is implied by this definition. Take the first form of Equation 16 and multiply by B from the left:
Equation:

$$\begin{aligned} & AB = \mathrm{I} \\ \Rightarrow \quad & B(AB) = \mathrm{BI} \\ \Rightarrow \quad & (BA)B = \mathrm{B} \\ \Rightarrow \quad & BA = \mathrm{I}. \end{aligned}$$

We emphasize that the inference made in the last step here is only valid when B and A are square matrices. The result means that, even though matrix multiplication is not commutative in general, we have a special case where it is commutative. If A and B are inverses of each other, then
Equation:

$$AB = \mathrm{BA} = \mathrm{I}.$$

Exercise:

Problem:

Prove that the inverse of the 2×2 rotation matrix $\mathrm{R}(\theta)$ is $\mathrm{R}^{T}(\theta)$.

Matrices that are not square do not have inverses. In fact, not all square matrices have inverses. So it becomes an important issue to determine which matrices do have inverses. If a matrix has an inverse, the inverse is unique. If a square matrix has no inverse, it is called a *singular* matrix. The *determinant* of a square matrix is a scalar computed from the numbers in the matrix. It tells us whether a matrix will have an inverse or not. A matrix is singular if and only if its determinant is zero.[footnote] In MATLAB, the notation `det(A)` is used to compute the determinant. Whenever the matrix A in Equation 4 is singular, it means one of two things about the system of

equations represented: either the equations are inconsistent and there is no solution, or the equations are dependent and there are infinitely many solutions.
It is not important now to understand how the determinant is defined and computed from the elements of a matrix. In your more advanced courses you will study the determinant in some detail.

Solving $Ax = \mathrm{b}$. Let's now study ways to solve the matrix equation $Ax = \mathrm{b}$. We will assume that a unique solution for x exists. Thus a unique matrix A^{-1} exists with the property $\mathrm{A}^{-1}\mathrm{A} = \mathrm{I}$. The trick is to find it. Here is one procedure.

For convenience, rewrite the matrix equation $\mathrm{Ax} = \mathrm{b}$ as
Equation:

$$\begin{bmatrix} A\mathrm{b} \end{bmatrix} \begin{bmatrix} \mathrm{x} \\ -1 \end{bmatrix} = 0.$$

The matrix $\begin{bmatrix} Ab \end{bmatrix} \in \mathscr{R}^{n\mathrm{x}(n+1)}$ is called the *augmented matrix* for the system of equations. The augmented matrix may be viewed as a systematic way of writing all the information necessary to solve the equations.

The advantage of Equation 19 is that we may premultiply both sides by any matrix C_1 (of compatible size), and the right-hand side remains zero (although this is equivalent to multiplying on both sides of Equation 4, which some may prefer). We can repeat this operation as often as we like with matrices $\mathrm{C}_2, \mathrm{C}_3$, etc. The general result is
Equation:

$$[\mathrm{C}_m \cdots \mathrm{C}_2\mathrm{C}_1\mathrm{A}\mathrm{C}_m \cdots \mathrm{C}_2\mathrm{C}_1\mathrm{b}] \begin{bmatrix} \mathrm{x} \\ -1 \end{bmatrix} = 0.$$

Now suppose that we have found a sequence of matrices $\mathrm{C}_1, \ldots, C_m$ that transforms the matrix A to the identity matrix:

$\mathrm{C}_m \cdots \mathrm{C}_2\mathrm{C}_1\mathrm{A} = \mathrm{I}$. The augmented matrix equation in 20 simplifies to

Equation:

$$\begin{bmatrix} I \mathrm{C}_m \cdots \mathrm{C}_2 \mathrm{C}_1 \mathrm{b} \end{bmatrix} \begin{bmatrix} \mathrm{x} \\ -1 \end{bmatrix} = 0,$$

which can be multiplied out to reveal that the solution for *x* is
Equation:

$$\mathrm{x} = \mathrm{C}_m \cdots \mathrm{C}_2 \mathrm{C}_1 \mathrm{b}.$$

The method may be summarized as follows:

i. form the augmented matrix $[Ab]$ for the system;
ii. premultiply $[Ab]$ by a sequence of selected matrices C_i, designed to take A to I; and
iii. when A is transformed to I, b will be transformed to x, so the solution will appear as the last column of the transformed augmented matrix.

We may also conclude that the product of the matrices C_i must be the inverse of A since A^{-1} is the unique matrix for which $\mathrm{A}^{-1}\mathrm{A} = \mathrm{I}$. In solving for *x* by this method, we have found A^{-1} implicitly as the product $\mathrm{C}_m \cdots \mathrm{C}_2 \mathrm{C}_1$.

Example:
Consider the equation
Equation:

$$\begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \end{bmatrix}$$

Equation:

$$A \ \mathrm{x} \ = \ \mathrm{b}.$$

The augmented matrix for this equation is

Equation:

$$[A\ \mathrm{b}]\ = \begin{bmatrix} 3 & 1 & 6 \\ 2 & 4 & 5 \end{bmatrix}.$$

Now if we could add $-2/3$ times the first row to the second row, we would get 0 in the lower left corner. This is the first step in transforming A to the identity I. We can accomplish this row operation with the matrix
Equation:

$$\mathrm{C}_1 = \begin{bmatrix} 1 & 0 \\ -2/3 & 1 \end{bmatrix}$$

Equation:

$$\mathrm{C}_1\ [A\ \mathrm{b}]\ = \begin{bmatrix} 3 & 1 & 6 \\ 0 & 10/3 & 1 \end{bmatrix}.$$

Now adding $-3/10$ times the new second row to the first row will introduce 0 in the $(1, 2)$ position, bringing us closer still to the identity. Thus
Equation:

$$\mathrm{C}_2 = \begin{bmatrix} 1 & -3/10 \\ 0 & 1 \end{bmatrix}$$

Equation:

$$\mathrm{C}_2\mathrm{C}_1\ [A\ \mathrm{b}]\ = \begin{bmatrix} 3 & 0 & 57/10 \\ 0 & 10/3 & 1 \end{bmatrix}.$$

We now complete the transformation to identity by normalizing each row to get the needed l's:
Equation:

$$\mathrm{C}_3 = \begin{bmatrix} 1/3 & 0 \\ 0 & 3/10 \end{bmatrix}$$

Equation:

$$\mathrm{C}_3\mathrm{C}_2\mathrm{C}_1 \ [A \ \mathrm{b}] \ = \begin{bmatrix} 1 & 0 & 19/10 \\ 0 & 1 & 3/10 \end{bmatrix}.$$

According to the last column, the solution is
Equation:

$$\mathrm{x} = \begin{bmatrix} 19/10 \\ 3/10 \end{bmatrix}.$$

We note in passing that the inverse of A is the product of the C's, so

$$\mathrm{A}^{-1} = \mathrm{C}_3\mathrm{C}_2\mathrm{C}_1 \ = \ \begin{bmatrix} 1/3 & 0 \\ 0 & 3/10 \end{bmatrix} \begin{bmatrix} 1 & -3/10 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -2/3 & 1 \end{bmatrix}$$

$$= \ \begin{bmatrix} 1/3 & -1/10 \\ 0 & 3/\mathrm{l}0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -2/3 & 1 \end{bmatrix} = \begin{bmatrix} 2/5 & -1/10 \\ -1/5 & 3/10 \end{bmatrix}.$$

The method we have just used, combined with a particular way of choosing the C_i matrices, is called *Gauss elimination*. Gauss elimination requires less computation than finding the inverse of A because only the effect of A^{-1} on the specific vector *b* is computed. MATLAB can solve for x by either method, as shown in Demo 4.2. For hand computations, we suggest choosing the C_i matrices so that C_1 produces O's everywhere below the diagonal in the first column, C_2 produces O's below the diagonal in the second column, and so on up to C_{n-1}. Then C_n produces O's above the diagonal in the nth column, C_{n+1} produces O's above the diagonal in column $n-1$, etc. The last one, C_{2n-1}, normalizes the diagonal elements to l's. We have assumed for simplicity that no O's on the diagonal will be encountered in hand computations.
Exercise:

Problem: Check that $\mathrm{A}^{-1}\mathrm{A} = \mathrm{I}$ in Example 1

Exercise:

Problem:

Augment Equation 3 as in Equation 19 and use the technique of Gauss elimination to solve for x.

Demo 1 (MATLAB). From the command level of MATLAB, solve the matrix equation of Example 1 from "Linear Algebra: Introduction" by typing

```
>> A = [1 1;0.3 -1.2]
>> b = [85;0]
```

You have entered the matrices A and *b*, which describe the problem. You can now solve for *x* by finding the inverse of *A* and multiplying *b*:

```
>> x = inv(A) * b
```

In this example the inverse is computed quickly because *A* is a small matrix. For a large (say, 30×30) matrix, the answer would take longer to compute, and the more efficient method of Gauss elimination would reduce waiting time. You can use Gauss elimination in MATLAB by typing

```
>> x = A \ b
```

You should get the same answer. Now type the MATLAB code required to compute Ax and to show $Ax - b = 0$.

Exercise:

Problem:

(MATLAB) Write the following system of equations as $Ax = \mathrm{b}$ and solve using MATLAB:

Equation:

$$3\left(x_1 - x_3\right) + 2\left(x_2 - 1\right) - 6 = x_3$$

Equation:

$$4x_3 = 7x_2 - 5$$

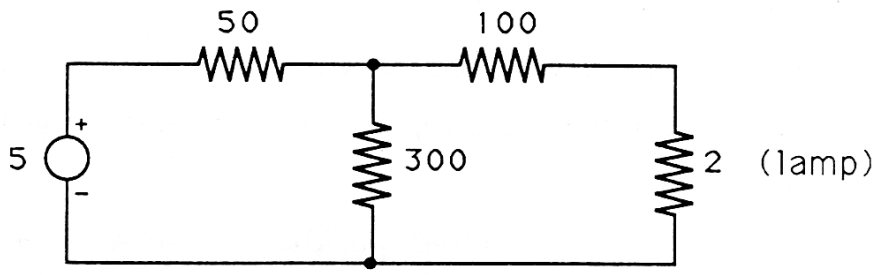
Equation:

$$6\left(x_1 + x_2 + 2\right) = x_3 + 1.$$

Linear Algebra: Circuit Analysis

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

In this section we use the linear algebra we have developed to find the voltages and currents in a simple electrical circuit, such as the one shown in Figure 1. There are many reasons why this might be necessary; in this example we need to know the current flowing through the lamp to tell whether or not it will glow. Too little current will have no visible effect; too much current will cause the lamp to burn out. We will apply a few physical laws relating the voltages and currents in a circuit, turn these laws into systems of linear equations, and solve the equations for the voltages and currents.



A Simple Electrical Circuit

Current, Voltage, and Resistance. We will use three physical quantities in our analysis of electrical circuits: current, voltage, and resistance. *Current* is the flow of electrical charge from one place to another. Electrons flowing through a wire or through some other electronic device comprise a current. *Voltage* is a difference in electric potential that makes electrons flow.

Voltage is sometimes called *electromotive force* because it is like a “force” that moves electrons. *Resistance* is a property of the device through which the electron current flows. The lower the resistance of a device, the more easily current can flow through the device.

The analogy of water flowing through pipes can help you develop intuition about electrical circuits. In this analogy, electrical current corresponds to the flow rate of water. Voltage corresponds to the pressure that forces the water to flow, and resistance is the friction of flow. A small pipe would impede the flow of water more than would a large pipe, so the small pipe would correspond to a higher resistance. While this analogy can be helpful, keep in mind that electricity is not water. All analogies break down at some point.

We measure electrical current in *amperes*. The standard symbol for current is i, and the direction of positive flow is indicated by an arrow on the circuit diagram. The arrow is for reference only; if the true current is in the opposite direction, we get negative values for i. Because electrons are negatively charged, current is defined as flowing in the opposite direction as electron motion. But to reduce confusion, you should learn to think in terms of current rather than electron motion.

A point in a circuit where several devices are connected together is called a *node*. The conservation law for current says that “what flows in must flow out of a node,” a principle known as *Kirchhoff's current law*. Kirchhoff's current law states that *the sum of all currents leaving a node is zero*. In this law, a current entering the node is considered to be a negative current leaving the node.

Voltage is measured in *volts* and is usually written as v (or e). Since voltage is a difference in potential between two points (nodes), we can show it on a circuit diagram with a $+$and a –sign to indicate which two nodes we are comparing and which one of the nodes is considered negative. As with current, the markings are for reference only and we may end up with a negative value of v.

In an electrical circuit, one node is usually chosen as a *reference node* and is considered to have a voltage of zero. Then the voltage at every other node is

measured with respect to the reference node. This saves us the trouble of always specifying pairs of nodes for voltage measurements and marking + and – signs for each voltage. Other names for the reference node are *common* and *ground*.

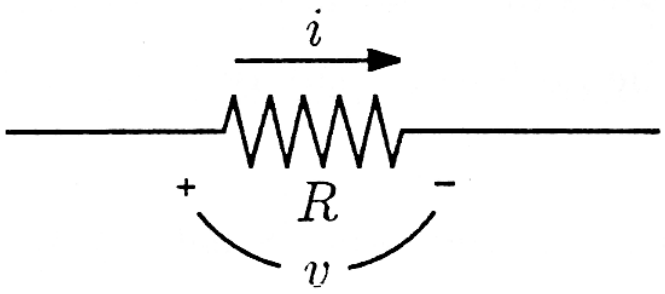
A *constant voltage source* is a device that always forces the voltage between its two terminals to be a constant value. In Figure 1 the circle at the left represents a constant voltage source of 5 volts, so that the voltage at the upper $(+)$ end is *always* exactly 5 volts higher than the voltage at the lower (-) end. A voltage source is something like a battery, but idealized. Real batteries do not maintain a constant output voltage under all conditions.

Resistance is measured in *ohms* and is denoted by R. A resistor is shown as a zig-zag line in circuit diagrams and labeled with the value of its resistance in ohms. In this chapter we will consider only devices whose resistance is positive and the same in both directions. *Ohm's law*, also called the *resistor law*, relates the voltage and current in a resistor. For the resistor shown in Figure 2, with reference directions assigned to v and i as shown, Ohm's law is

Equation:

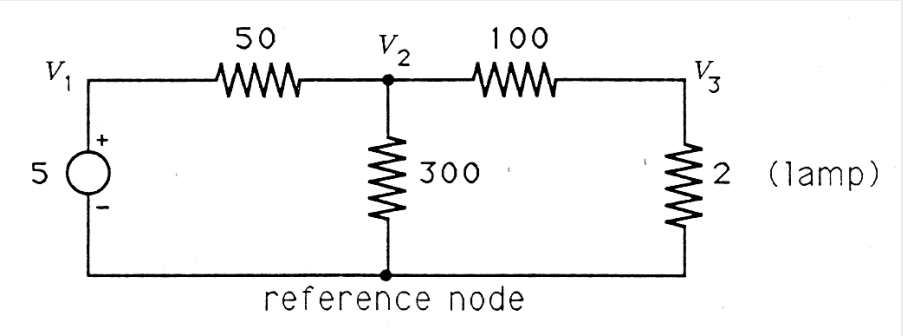
$$v = iR.$$

Note that current flows from + to - through the resistor.



Ohm's Law

Example:
Ohm's law and Kirchhoff's current law are the only principles we need to write equations that will allow us to find the voltages and currents in the resistive circuit of Figure 1. We begin by choosing a reference node and assigning variables to the voltages at every other node (with respect to the reference node). These choices are shown in Figure 3.



Assigning Node Voltages

The constant voltage source forces v_1 to be exactly 5 volts higher than the reference node. Thus
Equation:

$$v_1 = 5.$$

Next we write equations by applying Kirchhoff's current law to each node in the circuit (except the reference node and v_1, whose voltages we already know). At the node labeled v_2 are three paths for leaving current. The current leaving through the 50 ohm resistor can be found by Ohm's law, where the voltage across that resistor is $v_2 - v_1$:
Equation:

$$i_{50} = \frac{v}{R} = \frac{(v_2 - v_1)}{50}$$

For current leaving through the 300 ohm resistor, the voltage is v_2. Pay careful attention to the sign; since we are interested in the current *leaving* the node labeled v_2, Figure 4.14 indicates that to apply Ohm's law we should take the voltage as $+v_2-$ reference $= v_2 - 0 = v_2$. So

Equation:

$$i_{300} = \frac{v_2}{300}$$

For the 100 ohm resistor, we can write
Equation:

$$i_{100} = \frac{(v_2 - v_3)}{100}$$

According to Kirchhoff's current law, the sum of these three leaving currents is zero:
Equation:

$$\begin{aligned} & \tfrac{(v_2 - v_1)}{50} + \tfrac{v_2}{300} + \tfrac{(v_2 - v_3)}{100} = 0 \\ \Rightarrow \quad & 6\,(v_2 - v_1) + v_2 + 3\,(v_2 - v_3) = 0 \\ \Rightarrow \quad & -6v_1 + 10v_2 - 3v_3 = 0. \end{aligned}$$

Notice that when we wrote the equation for the node labeled v_2, the variable v_2 had a $+$ sign each time it occurred in the equation, while the others had a -sign. This is always the case, and watching for it can help you avoid sign errors. Now we apply Kirchhoff's current law at the node labeled v_3 to get the equation
Equation:

$$\begin{aligned} & \tfrac{(v_3 - v_2)}{100} + \tfrac{v_3}{2} = 0 \\ \Rightarrow \quad & (v_3 - v_2) + 50v_3 = 0 \\ \Rightarrow \quad & 0v_1 - 1v_2 + 51v_3 = 0. \end{aligned}$$

Note that this time it is v_3 that always shows up with a $+$ sign.
Equations 2, 6, and 7 give us a system of three equations in the three unknown variables v_1, v_2, and v_3. We now write them in matrix form as
Equation:

$$\begin{array}{rrrcrl} 1 & 0 & 0 & v_1 & = & 5 \\ -6 & 10 & -3 & v_2 & = & 0 \\ 0 & -1 & 51 & v_3 & = & 0 \end{array} .$$

Exercise:

Problem:

(MATLAB) Use MATLAB to solve Equation 8. You should find
Equation:

$$v_1 = 5.0000 \ volts$$

Equation:

$$v_2 = 3.0178 \ volts$$

Equation:

$$v_3 = 0.0592 \ volt.$$

What is the determinant of the coefficient matrix A? Is the solution unique?

We can determine the current flowing through the lamp from v_3 to ground in Example 1 by Ohm's law:
Equation:

$$i = \frac{v}{R} = \frac{v_3}{2} = 0.0296 \ \text{ampere}.$$

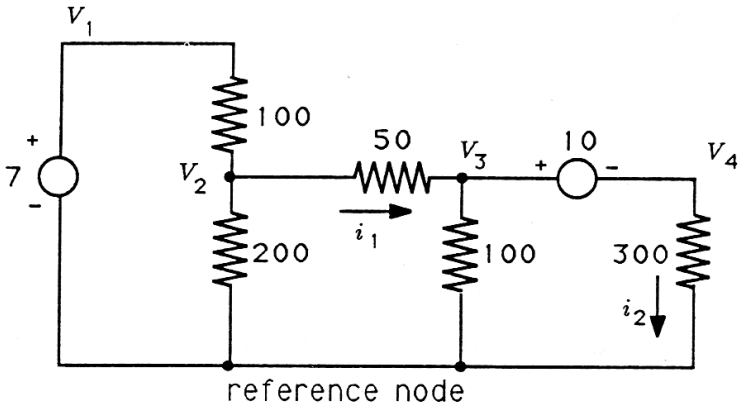
The visible effect will, of course, depend on the lamp. Let us assume that the specifications for our lamp indicate that 0.05 ampere or more is required

before it will glow, and more than 0.075 ampere will cause it to burn out. In this case, our circuit would not make the lamp glow.

Exercise:

Problem:

(MATLAB) Write and solve equations for the circuit in Figure 4. What are the voltages at the nodes labeled v_1 through v_4? What is the current labeled i_1 ? And i_2?



A Resistive Network

Linear Algebra: Numerical Experiment (Circuit Design)

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Our analysis in Example 1 from "Linear Algebra: Circuit Analysis" and Problem 1 from "Linear Algebra: Circuit Analysis" indicates that not enough current will flow through the lamp to make it glow. We now wish to change the resistance of the 100 ohm resistor to a new value so that the lamp will glow. We replace 100 in the equations with an unknown resistance R. Equation 2 is unchanged, but Equation 6 becomes
Equation:

$$\begin{aligned} & \tfrac{(v_2-v_1)}{50}+\tfrac{v_2}{300}+\tfrac{(v_2-v_3)}{R}=0 \\ \Rightarrow \quad & 6R\left(v_2-v_1\right)+Rv_2+300\left(v_2-v_3\right)=0 \\ \Rightarrow \quad & -6Rv_1+(7R+300)v_2-300v_3=0. \end{aligned}$$

Equation 7 from "Linear Algebra: Circuit Analysis" becomes
Equation:

$$\begin{aligned} & \tfrac{(v_3-v_2)}{R}+\tfrac{v_3}{2}=0 \\ \Rightarrow \quad & 2\left(v_3-v_2\right)+Rv_3=0 \\ \Rightarrow \quad & 0v_1-2v_2+(R+2)v_3=0. \end{aligned}$$

The matrix form of these equations is
Equation:

$$\begin{matrix} 1 & & 0 & 0 & v_1 & & 5 \\ -6R & 7R & +300 & -300 & v_2 & = & 0 \\ 0 & & -2 & R+2 & v_3 & & 0 \end{matrix} .$$

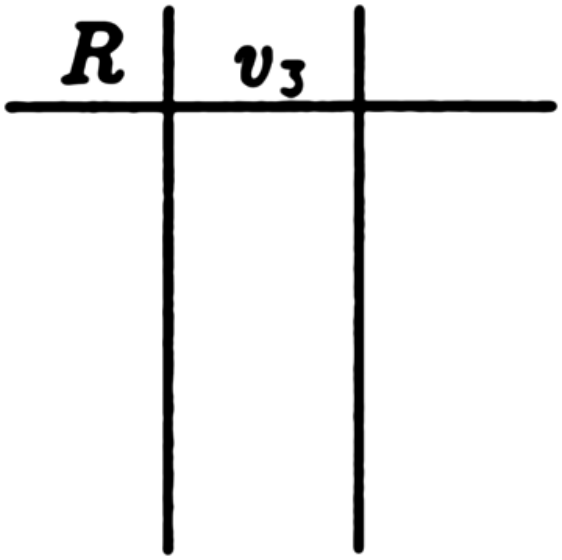
Write a MATLAB function file called `builda` to accept R as an input and return the matrix A in Equation 3 as an output. The first line of your function file should be

```
function A = builda(R);
```

Now choose several values for R. For each choice, use your function *builda* and solve the resulting matrix equation $Av = \mathrm{b}$ for the voltages. Each time you choose a different R to build a different matrix A, check the determinant of A to be sure the equations have a unique solution:

```
>> det(A)
```

Make a table of R and the corresponding values of v_3:



Now add a column to your table for the current through the lamp $i = v_3/2$. Add rows to your table until you have found a value of R for which the lamp will glow. (i needs to be between 0.05 and 0.075 ampere.)

Vector Graphics: Introduction

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Acknowledgements
Fundamentals of Interactive Computer Graphics by J. D. Foley and A. Van Dam, ©1982 Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, was used extensively as a reference book during development of this chapter. Star locations were obtained from the share-ware program “Deep Space” by David Chandler, who obtained them from the “Skymap” database of the National Space Science Data Center.

Notes to Teachers and Students:

In this chapter we introduce matrix data structures that may be used to represent two- and three-dimensional images. The demonstration program shows students how to create a function file for creating images from these data structures. We then show how to use matrix transformations for translating, scaling, and rotating images. Projections are used to project three-dimensional images onto two-dimensional planes placed at arbitrary locations. It is precisely such projections that we use to get perspective drawings on a two-dimensional surface of three-dimensional objects. The numerical experiment encourages students to manipulate a star field and view it from several points in space.

Once again we consider certain problems essential to the chapter development. For this chapter be sure not to miss the following exercises: Exercise 2 in "Two-Dimensional Image Transformations", Exercise 1 in "Homogeneous Coordinates", Exercise 2 in "Homogeneous Coordinates",

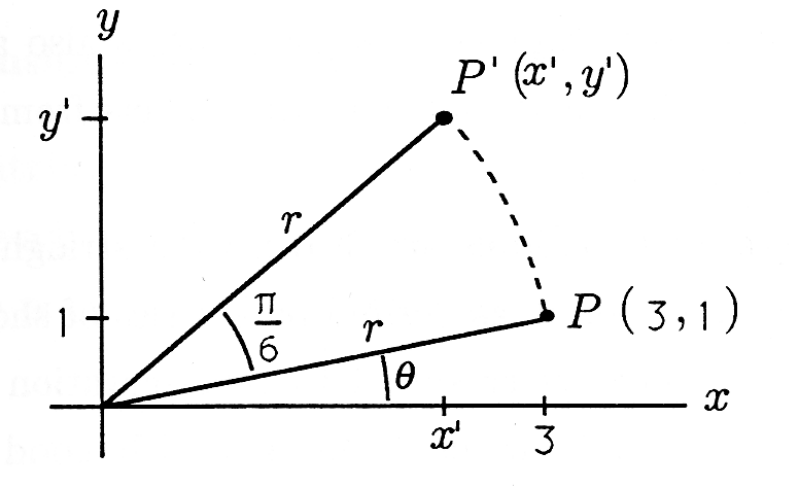
Exercise 5 in "Three-Dimensional Homogeneous Coordinates", and Exercise 2 in "Projections".

Introduction

Pictures play a vital role in human communication, in robotic manufacturing, and in digital imaging. In a typical application of digital imaging, a CCD camera records a digital picture frame that is read into the memory of a digital computer. The digital computer then manipulates this frame (or array) of data in order to crop, enlarge or reduce, enhance or smooth, translateor rotate the original picture. These procedures are called *digital picture processing* or *computer graphics*. When a sequence of picture frames is processed and displayed at video frame rates (30 frames per second), then we have an animated picture.

In this chapter we use the linear algebra we developed in The chapter on Linear Algebra to develop a rudimentary set of tools for doing computer graphics on line drawings. We begin with an example: the rotation of a single point in the (x, y) plane.

Example:
Point P has coordinates $(3, 1)$ in the (x, y) plane as shown in Figure 1. Find the coordinates of the point $P\prime$, which is rotated $\frac{\pi}{6}$ radians from P.



Rotating a Single Point in the (x, y) Plane

To solve this problem, we can begin by converting the point P from rectangular coordinates to polar coordinates. We have
Equation:

$$r = \sqrt{x^2 + y^2} = \sqrt{10}$$
$$\theta = \tan^{-1}\left(\frac{y}{x}\right) \approx 0.3217 \ radian.$$

The rotated point $P\prime$ has the same radius r, and its angle is $\theta + \frac{\pi}{6}$. We now convert back to rectangular coordinates to find $x\prime$ and $y\prime$ for point $P\prime$:
Equation:

$$x' = r \ cos \ \left(\theta + \tfrac{\pi}{6}\right) \approx \sqrt{10} \cos\left(0.8453\right) \approx 2.10$$
$$y' = r \ sin \ \left(\theta + \tfrac{\pi}{6}\right) \approx \sqrt{10} \sin\left(0.8453\right) \approx 2.37.$$

So the rotated point $P\prime$ has coordinates (2.10, 2.37).

Now imagine trying to rotate the graphical image of some complex object like an airplane. You could try to rotate all 10,000 (or so) points in the same way as the single point was just rotated. However, a much easier way to rotate all the points together is provided by linear algebra. In fact, with a single linear algebraic operation we can rotate and scale an entire object and project it from three dimensions to two for display on a flat screen or sheet of paper.

In this chapter we study *vector graphics*, a linear algebraic method of storing and manipulating computer images. Vector graphics is especially suited to moving, rotating, and scaling (enlarging and reducing) images and objects within images. Cropping is often necessary too, although it is a little more difficult with vector graphics. Vector graphics also allows us to store objects in three dimensions and then view the objects from various locations in space by using projections.

In vector graphics, pictures are drawn from straight lines.[footnote] A curve can be approximated as closely as desired by a series of short, straight lines. Clearly some pictures are better suited to representation by straight lines than are others. For example, we can achieve a fairly good representation of a building or an airplane in vector graphics, while a photograph of a forest would be extremely difficult to convert to straight lines. Many computer-aided design (CAD) programs use vector graphics to manipulate mechanical drawings.
It is possible to extend these techniques to deal with some types of curves, but we will consider only straight lines for the sake of simplicity.

When the time comes to actually display a vector graphics image, it may be necessary to alter the representation to match the display device. Personal computer display screens are divided into thousands of tiny rectangles called *picture elements*, or *pixels*. Each pixel is either off (black) or on (perhaps with variable intensity and/or color). With a CRT display, the electron beam scans the rows of pixels in a raster pattern. To draw a line on a pixel display device, we must first convert the line into a list of pixels to be illuminated. Dot matrix and laser printers are also pixel display devices, while pen plotters and a few specialized CRT devices can display vector

graphics directly. We will let MATLAB do the conversion to pixels and automatically handle cropping when necessary.

We begin our study of vector graphics by representing each point in an image by a vector. These vectors are arranged side-by-side into a matrix G containing all the points in the image. Other matrices will be used as operators to perform the desired transformations on the image points. For example, we will find a matrix R, which functions as a rotation: the matrix product RG represents a rotated version of the original image G.

Vector Graphics: Two-Dimensional Image Representation

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Point Matrix. To represent a straight-line image in computer memory, we must store a list of all the endpoints of the line segments that comprise the image. If the point $P_i = (x_i, y_i)$ is such an endpoint, we write it as the column vector

Equation:

$$\mathrm{p}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Suppose there are n such endpoints in the entire image. Each point is included only once, even if several lines end at the same point. We can arrange the vectors P_i into a point matrix:

Equation:

$$\begin{aligned} \mathrm{G} &= [\mathrm{p}_1 \mathrm{p}_2 \mathrm{p}_3 \ldots \mathrm{p}_n] \\ &= \begin{bmatrix} x_1 & x_2 & x_3 & x_n \\ y_1 & y_2 & y_3 & y_n \end{bmatrix}. \end{aligned}$$

We then store the point matrix $\mathrm{G} \in \mathscr{R}^{2\mathrm{x}n}$ as a two-dimensional array in computer memory.

Example:

Consider the list of points
Equation:

$$P_1 = (0, 0)$$
$$P_2 = (-1.5, 5)$$
$$P_3 = (4, 2.3)$$
$$P_4 = (4, -1).$$

The corresponding point matrix is
Equation:

$$\mathrm{G} = \begin{bmatrix} 0 & -1.5 & 4 & 4 \\ 0 & 5 & 2.3 & -1 \end{bmatrix}.$$

Line Matrix. The next thing we need to know is which pairs of points to connect with lines. To store this information for m lines, we will use a line matrix, $\mathrm{H} \in \mathscr{R}^{2 \times m}$. The line matrix *does not* store line locations directly. Rather, it contains references to the points stored in G. To indicate a line between points p_i and p_j, we store the indices i and j as a pair. For the kth line in the image, we have the pair
Equation:

$$\mathrm{h}_k = \begin{bmatrix} i_k \\ j_k \end{bmatrix}.$$

The order of i and j does not really matter since a line from p_i to p_j is the same as a line from p_j to p_i. Next we collect all the lines h_k into a line matrix H:
Equation:

$$\mathrm{H} = \begin{bmatrix} i_1 & i_2 & i_3 & \dots & i_m \\ j_1 & j_2 & j_3 & \dots & j_m \end{bmatrix}.$$

All the numbers in the line matrix H will be positive integers since they point to columns of G. To find the actual endpoints of a line, we look at columns i and j of the point matrix G.

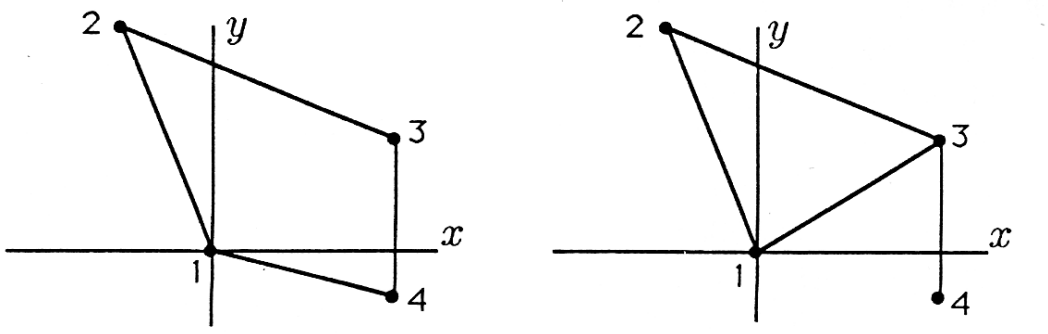
Example:
To specify line segments connecting the four points of Example 1 into a quadrilateral, we use the line matrix
Equation:

$$\mathrm{H}_1 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{bmatrix}.$$

Alternatively, we can specify line segments to form a triangle from the first three points plus a line from P_3 to P_4 :
Equation:

$$\mathrm{H}_2 = \begin{bmatrix} 1 & 2 & 3 & 3 \\ 2 & 3 & 1 & 4 \end{bmatrix}.$$

Figure 1 shows the points G connected first by H_1 and then by H_2.



Two Sets of Lines

Demo 1 (MATLAB). Use your editor to enter the following MATLAB function file. Save it as vgraphl.m.

```
function vgraphl (points, lines);
% vgraphl (points, lines) plots the points as *'s
and
% connects the points with specified lines. The po
ints
% matrix should be 2xN, and the lines matrix shoul
d be 2xM.
% The field of view is preset to (-50,50) on both
axes.
%
% Written by Richard T. Behrens, October 1989
%
m=length(lines);                    % find the num
ber of
                                    % lines.
axis([-50 50 -50 50])               % set the axis
 scales
axis('square')
plot(points(1,:),points(2,:),'*')   % plot the poi
nts as *
hold on                             % keep the poi
nts...
for i=i:m                           % while plotti
ng the
                                    % lines
    plot([points(1,lines(1,i)) points(1,lines(2,i)
)],..
        [points(2,lines(2,lines(1,i)) points(2,lin
es(2,i))],'-')

    end

    hold off
```

After you have saved the function file, run MATLAB and type the following to enter the point and line matrices. (We begin with the transposes of the matrices to make them easier to enter.)

```
>> G = [
    0.6052 -0.4728;
   -0.4366  3.5555;
   -2.6644  7.9629;
   -7.2541 10.7547;
  -12.5091 11.5633;
  -12.5895 15.1372;
   -6.5602 13.7536;
  -31.2815 -7.7994;
  -38.8314 -9.9874;
  -44.0593 -1.1537;
  -38.8314  2.5453;
  -39.4017  9.4594;
  -39.3192 15.0932;
  -45.9561 23.4158]

>> G = G'

>> H = [
   1   2;
   2   3;
   3   4;
   4   5;
   4   7;
   5   6;
   8   9;
   9   10;
   10  11;
   11  12;
   12  13;
   12  14]

>> H = H'
```

At this point you should use MATLAB's "save" command to save these matrices to a disk file. Type

```
>> save dippers
```

After you have saved the matrices, use the function `VGRAPH1` to draw the image by typing

```
≫ vgraph1(G,H)
```

The advantage of storing points and lines separately is that an object can be moved and scaled by operating only on the point matrix G. The line information in H remains the same since the same pairs of points are connected no matter where we put the points themselves.

Surfaces and Objects. To describe a surface in three dimensions is a fairly complex task, especially if the surface is curved. For this reason, we will be satisfied with points and lines, sometimes visualizing flat surfaces based on the lines. On the other hand, it is a fairly simple matter to group the points and lines into distinct *objects*. We can define an object matrix K with one column for each object giving the ranges of points and lines associated with that object. Each column is defined as

Equation:

$$\mathrm{k}_i = \begin{matrix} \text{first} & \text{point} \\ \text{last} & \text{point} \\ \text{first} & \text{line} \\ \text{last} & \text{line} \end{matrix} .$$

As with the line matrix H, the elements of K are integers.

Example:
Consider again Demo 1. We could group the points in G and the lines in H into two objects with the matrix

Equation:

$$\mathrm{K} = \begin{matrix} 1 & 8 \\ 7 & \mathrm{l}4 \\ 1 & 7 \\ 6 & 12 \end{matrix} .$$

The first column of K specifies that the first object (Ursa Minor) is made up of points 1 through 7 and lines 1 through 6, and the second column of K defines the second object (Ursa Major) as points 8 through 14 and lines 7 through 12.

Vector Graphics: Two-Dimensional Image Transformations

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

We now turn our attention to operating on the point matrix G to produce the desired transformations. We will consider rotation scaling and translation (moving) of objects. Rotation and scaling are done by matrix multiplication with a square transformation matrix A. If we call the transformed point matrix $\mathrm{G}_{\mathrm{new}}$, we have
Equation:

$$[\mathrm{G}_{\mathrm{new}}] = [\mathrm{A}]\,[\mathrm{G}].$$

We call A a *matrix operator* because it "operates" on G through matrix multiplication. In contrast, translation must be done by matrix addition.

In a later section you will see that it is advantageous to perform all operations by matrix operators and that we can modify our image representation to allow translation to be done with a matrix operator like rotation and scaling. We will call the modified representation *homogeneous coordinates*.

Rotation. We saw in the chapter on linear algebra that the matrix that rotates points by an angle θ is
Equation:

$$\mathrm{A} = \mathrm{R}\,(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

When applied to the point matrix G, this matrix operator rotates each point by the angle θ, regardless of the number of points.

Example:
We can use the rotation matrix to do the single point rotation of the example from "Vector Graphics: Introduction". We have a point matrix consisting of only the point $(3, 1)$:
Equation:

$$\mathrm{G} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

The necessary transformation matrix is $\mathrm{R}(\theta)$ with $\theta = \dfrac{\pi}{6}$ Then the rotated point is given by
Equation:

$$\mathrm{G}_{\text{new}} = \mathrm{R}\left(\frac{\pi}{6}\right)\mathrm{G} = \begin{matrix} \cos\ \frac{\pi}{6} & -\sin\ \frac{\pi}{6} \\ \sin\ \frac{\pi}{6} & \cos\ \frac{\pi}{6} \end{matrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix} \approx \begin{bmatrix} 2.10 \\ 2.37 \end{bmatrix}.$$

Scaling. An object can be enlarged or reduced in each dimension independently. The matrix operator that scales an image by a factor of s_x along the x-axis and s_y along the y-axis is
Equation:

$$\mathrm{A} = \mathrm{S}\left(s_x, s_y\right) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}.$$

Most often we take $s_x = s_y$ to scale an image by the same amount in both dimensions.
Exercise:

Problem:

Write out the following matrices. Simplify and give numerical answers to two decimal places:

a. $\mathrm{R}\left(\frac{\pi}{2}\right)$;
b. $\mathrm{S}(3,2)$;
c. $\mathrm{R}\left(-\frac{\pi}{4}\right)$;
d. $\mathrm{S}(-1,1)$.

Exercise:
Reflections

Problem: What does $\mathrm{S}(-1,1)$ do? $\mathrm{S}(1,-1)$?$\mathrm{S}(-1,-1)$? $\mathrm{S}(1,1)$?

Exercise:

Problem:

Given $\mathrm{G}=\begin{bmatrix}0 & -1.5 & 4 & 4\\ 0 & 5 & 2.3 & -1\end{bmatrix}$ and $\theta=\frac{\pi}{3}$ find $\mathrm{G}_{\text{new}}=\mathrm{R}(\theta)\mathrm{G}$. Give numerical answers to two decimal places.

Exercise:

Problem:

Apply each of the transformations in Exercise 1 and 2 to the image
Equation:

$$\mathrm{G}=\begin{bmatrix}1 & 1 & 2 & 2\\ 1 & 2 & 2 & 1\end{bmatrix}\ ;\ \mathrm{H}=\begin{bmatrix}1 & 2 & 3 & 4\\ 2 & 3 & 4 & 1\end{bmatrix}.$$

Sketch the original image and each transformation of it.

Translation. An object can be moved by adding a constant vector *b* to every point in the object. For example, $\mathrm{b} = \begin{bmatrix} 20 \\ -5 \end{bmatrix}$ will move an object 20 units to the right and 5 units down. We can write this in terms of the point matrix as

Equation:

$$\mathrm{G}_{\mathrm{new}} = \mathrm{G} + \mathrm{b}1^T$$

where 1 (read “the one-vector”) is a vector of *n* l's:

Equation:

$$1 = \begin{matrix} 1 \\ 1 \\ \vdots \\ 1 \end{matrix} .$$

In MATLAB, 1 may be obtained by `ones(n,1)`. The outer product of b with 1 in Equation 7 simply serves to make n copies of b so that one copy can be added to each point in G.

Vector Graphics: Composition of Transformations

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Often we will want to perform several operations on an object before we display the result. For example, suppose we want to rotate by $\frac{\pi}{3}$ and reduce to $\frac{1}{2}$ size in each dimension:
Equation:

$$\mathrm{G}_1 = \mathrm{R}\left(\frac{\pi}{3}\right)\mathrm{G}$$
$$\mathrm{G}_{\text{new}} = \mathrm{S}\left(\tfrac{1}{2}, \tfrac{1}{2}\right)\mathrm{G}_1.$$

If there are n points in the matrix G, it will require $4n$ multiplications to perform each of these operations, for a total of $8n$ multiplications. However, we can save some multiplications by noting that
Equation:

$$\mathrm{G}_{\text{new}} = \mathrm{S}\left(\frac{1}{2}, \frac{1}{2}\right)\left[\mathrm{R}\left(\frac{\pi}{3}\right)\mathrm{G}\right] = \mathrm{AG}$$

where
Equation:

$$\begin{aligned} \mathrm{A} &= \mathrm{S}\left(\tfrac{1}{2}, \tfrac{1}{2}\right)\mathrm{R}\left(\tfrac{\pi}{3}\right) \\ &= \begin{bmatrix} \frac{1}{2}\cos\left(\frac{\pi}{3}\right) & -\frac{1}{2}\sin\left(\frac{\pi}{3}\right) \\ \frac{1}{2}\sin\left(\frac{\pi}{3}\right) & \frac{1}{2}\cos\left(\frac{\pi}{3}\right) \end{bmatrix}. \end{aligned}$$

In other words, we take advantage of the fact that matrix multiplication is associative to combine S and R into a single operation A, which requires only 8 multiplications. Then we operate on G with A, which requires $4n$ multiplications. By "composing" the two operations, we have reduced the total from $8n$ to $4n + 8$ multiplications. Furthermore, we can now build operators with complex actions by combining simple actions.

Example:
We can build an operator that stretches objects along a diagonal line by composing scaling and rotation. We must

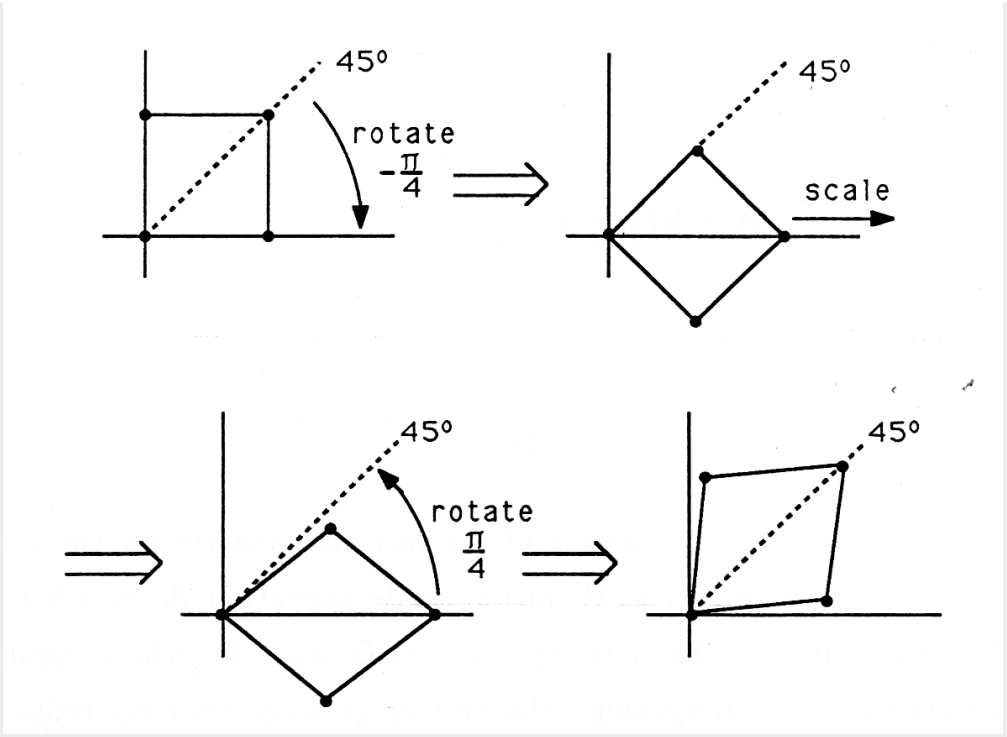
i. rotate the diagonal line to the x-axis with $\mathrm{R}(-\theta)$;
ii. scale with $\mathrm{S}(s, 1)$; and
iii. rotate back to the original orientation with $\mathrm{R}(\theta)$.

Figure 1 shows a square being stretched along a *45º* line. The composite operator that performs this directional stretching is
Equation:

$$\begin{aligned} \mathrm{A}(\theta, s) &= \mathrm{R}(\theta)\mathrm{S}(s, 1)\mathrm{R}(-\theta) \\ &= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} s & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} s\cos^2\theta + \sin^2\theta & (s-1)\sin\theta\cos\theta \\ (s-1)\sin\theta\cos\theta & \cos^2\theta + s\sin^2\theta \end{bmatrix}. \end{aligned}$$

Note that the rightmost operator in a product of operators is applied first.

Rotating and Scaling for Directional Stretching


45°
rotate
$-\frac{\pi}{4}$
45°
scale
45°
rotate
$\frac{\pi}{4}$
45°

Vector Graphics: Homogeneous Coordinates

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

In the previous section we were able to combine rotation and scaling into a single composite operation by matrix multiplication. Unfortunately, translation cannot yet be included in the composite operator since we do it by addition rather than by multiplication.

Example:
Suppose we wish to rotate the image G by $\frac{\pi}{3}$ about the point $(-10, 10)$. Our rotation matrix $\mathrm{R}(\theta)$ always rotates about the origin, so we must combine three transformations to accomplish this:

i. translate the point $(-10, 10)$ to the origin;
ii. rotate $\frac{\pi}{3}$ radians about the origin; and
iii. translate the origin back to $(-10, 10)$.

For step (i), we have $\mathrm{b}_0 = \begin{bmatrix} 10 \\ -10 \end{bmatrix}$ and

Equation:

$$\mathrm{G}_1 = \mathrm{G} + \mathrm{b}_0 1^T.$$

For step (ii),
Equation:

$$\begin{aligned} \mathrm{G}_2 &= \mathrm{R}\left(\tfrac{\pi}{3}\right)\mathrm{G}_1 \\ &= \mathrm{R}\left(\tfrac{\pi}{3}\right)\left[\mathrm{G} + \mathrm{b}_0 1^T\right] \\ &= \mathrm{R}\left(\tfrac{\pi}{3}\right)\mathrm{G} + \mathrm{R}\left(\tfrac{\pi}{3}\right)\mathrm{b}_0 1^T. \end{aligned}$$

For step (iii), we can use $-\mathrm{b}_0$ from step (i):
Equation:

$$\begin{aligned} \mathrm{G}_{\text{new}} &= \mathrm{G}_2 - \mathrm{b}_0 1^T \\ &= \mathrm{R}\left(\tfrac{\pi}{3}\right)\mathrm{G} + \mathrm{R}\left(\tfrac{\pi}{3}\right)\mathrm{b}_0 1^T - \mathrm{b}_0 1^T \\ &= \mathrm{R}\left(\tfrac{\pi}{3}\right)\mathrm{G} + \left[\left(\mathrm{R}\left(\tfrac{\pi}{3}\right) - \mathrm{I}\right)\mathrm{b}_0\right] 1^T. \end{aligned}$$

In this example we were unable to find a single matrix operator A to do the entire job. The total transformation took the form

Equation:

$$\mathrm{G}_{\text{new}} = \mathrm{A}\mathrm{G} + \mathrm{b} 1^T.$$

This is called an *affine transformation* because it involves both multiplication by A and addition of a constant matrix. This is in contrast to the more desirable *linear transformation*, which involves only multiplication by A.

We will now move toward a modified representation of the image and the operators by rewriting the last equation as
Equation:

$$\mathrm{G}_{\text{new}} = [\mathrm{A}\mathrm{b}] \begin{bmatrix} \mathrm{G} \\ 1^T \end{bmatrix}$$

where in the example we had $\mathrm{A} = \mathrm{R}\left(\dfrac{\pi}{3}\right)$ and $\mathrm{b} = \left(\mathrm{R}\left(\dfrac{\pi}{3}\right) - \mathrm{I}\right)\mathrm{b}_0$.

Exercise:

Problem: Show that, for any matrices $\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}$ of compatible sizes,
Equation:

$$\mathrm{AB} + \mathrm{CD} = \begin{bmatrix} A & \mathrm{C} \end{bmatrix} \begin{bmatrix} \mathrm{B} \\ \mathrm{D} \end{bmatrix}.$$

The matrix $\begin{bmatrix} \mathrm{G} \\ 1^T \end{bmatrix}$ looks like
Equation:

$$\begin{matrix} x_1 & x_2 & x_n \\ y_1 & y_2 & y_n \\ 1 & 1 & 1 \end{matrix},$$

and the points $(x_i, y_i, 1)$ are called *homogeneous coordinates*. We can modify Equation 5 so that the new point matrix is also in homogeneous coordinates:
Equation:

$$\begin{bmatrix} \mathrm{G}_{\mathrm{new}} \\ 1^T \end{bmatrix} = \begin{bmatrix} \mathrm{A} & \mathrm{b} \\ \mathrm{O}^T & 1 \end{bmatrix} \begin{bmatrix} \mathrm{G} \\ 1^T \end{bmatrix}.$$

In the new representation, each point in the image has a third coordinate, which is always a 1. The homogeneous transformation is a 3×3 matrix,
Equation:

$$\mathrm{A}_h = \begin{bmatrix} \mathrm{A} & \mathrm{b} \\ 0^T & 1 \end{bmatrix},$$

which is capable of translation, rotation, and scaling all by matrix multiplication. Thus, using homogeneous coordinates, we can build composite

transformations that include translation.

In homogeneous coordinates, we have
Equation:

$$\mathrm{R}(\theta) = \begin{matrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{matrix}$$

$$\mathrm{S}(s_x, s_y) = \begin{matrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{matrix}$$

$$\mathrm{T}(t_x, t_y) = \begin{matrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{matrix} .$$

Example:
The composite transformation to triple the size of an image and *then* move it 2 units to the left is
Equation:

$$\mathrm{A} = \begin{matrix} 1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & \mathrm{l} \end{matrix} \begin{matrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{matrix} = \begin{matrix} 3 & 0 & -2 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{matrix} .$$

On the other hand, the composite transformation to move an image 2 units to the left and *then* triple its size is
Equation:

$$\mathrm{B} = \begin{matrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{matrix} \begin{matrix} 1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} = \begin{matrix} 3 & 0 & -6 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{matrix} .$$

In the latter case, the distance of the translation is also tripled.

Exercise:

Problem:

Find a single composite transformation in homogeneous coordinates that rotates an image by angle θ about point (x_i, y_i) as in Example 1.

Vector Graphics: Three-Dimensional Homogeneous Coordinates

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

We now consider the storage and manipulation of three-dimensional objects. We will continue to use homogeneous coordinates so that translation can be included in composite operators. Homogeneous coordinates in three dimensions will also allow us to do perspective projections so that we can view a three-dimensional object from any point in space.

Image Representation. The three-dimensional form of the point matrix in homogeneous coordinates is
Equation:

$$\mathrm{G} = \begin{matrix} x_1 & x_2 & x_3 & \ldots & x_n \\ y_1 & y_2 & y_3 & \cdots & y_n \\ z_1 & z_2 & z_3 & \ldots & z_n \\ 1 & 1 & 1 & \ldots & 1 \end{matrix} \quad \in \mathscr{R}^{4\times n}.$$

The line matrix H is exactly as before, pointing to pairs of columns in G to connect with lines. Any other grouping matrices for objects, etc., are also unchanged.

Image manipulations are done by a 4×4 matrix A. To ensure that the fourth coordinate remains a 1, the operator A must have the structure
Equation:

$$A = \begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{matrix}.$$

The new image has point matrix
Equation:

$$\mathrm{G}_{\mathrm{new}} = AG.$$

Exercise:

Problem:

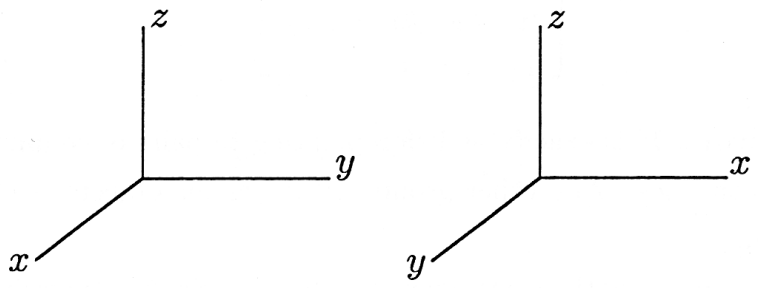
If the coordinates of the ith point in G are $(x_i, y_i, z_i, 1)$, what are the coordinates of the ith point in $\mathrm{G}_{\mathrm{new}} = \mathrm{AG}$ when A is as given in Equation 2?

Exercise:

Problem:

Write down the point matrix G for the unit cube (the cube with sides of length 1, with one corner at the origin and extending in the positive direction along each axis). Draw a sketch of the cube, numbering the vertices according to their order in your point matrix. Now write down the line matrix H to complete the representation of the cube.

Left- and Right-Handed Coordinate Systems. In this book we work exclusively with *right-handed* coordinate systems. However, it is worth pointing out that there are two ways to arrange the axes in three dimensions. Figure 1(a) shows the usual *right-handed* coordinates, and the *left-handed* variation is shown in Figure 1(b). All possible choices of labels x, y, and z for the three coordinate axes can be rotated to fit one of these two figures, but no rotation will go from one to the other.



Three-Dimensional Coordinate Systems; (a) Right-handed, and (b) Left-handed

Be careful to sketch a right-handed coordinate system when you are solving problems in this chapter. Some answers will not be the same for a left-handed system.

Image Transformation. Three-dimensional operations are a little more complicated than their two-dimensional counterparts. For scaling and translation we now have three independent directions, so we generalize the operators of Equation 10 from "Vector Graphics: Homogeneous Coordinates" as

Equation:

$$\mathrm{S}(s_x, s_y, s_z) = \begin{matrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$\mathrm{T}(t_x, t_y, t_z) = \begin{matrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{matrix} .$$

Exercise:

Problem:

Show that $\mathrm{T}(-t_x, -t_y, -t_z)$ is the inverse of $\mathrm{T}\left(t_x, t_y, t_z\right)$. T^{-1} undoes the work of T.

Rotation may be done about any arbitrary line in three dimensions. We will build up to the general case by first presenting the operators that rotate about the three coordinate axes. $\mathrm{R}_x\left(\theta\right)$ rotates by angle θ about the x-axis, with positive θ going from the y-axis to the z-axis, as shown in Figure 2. In a similar fashion, positive rotation about the y-axis using $\mathrm{R}_y\left(\theta\right)$ goes from z to x, and positive rotation about the z-axis goes from x to y, just as in two dimensions. We have the fundamental rotations

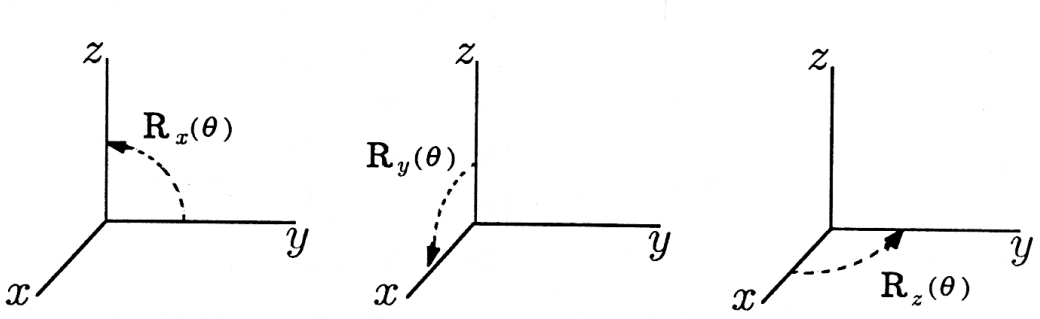
Equation:

$$\mathrm{R}_x\left(\theta\right) = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$\mathrm{R}_y\left(\theta\right) = \begin{matrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$\mathrm{R}_z\left(\theta\right) = \begin{matrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} .$$

A more general rotation about any line through the origin can be constructed by composition of the three fundamental rotations. Finally, by composing translation with the fundamental rotations, we can build an operator that rotates about any arbitrary line in space.



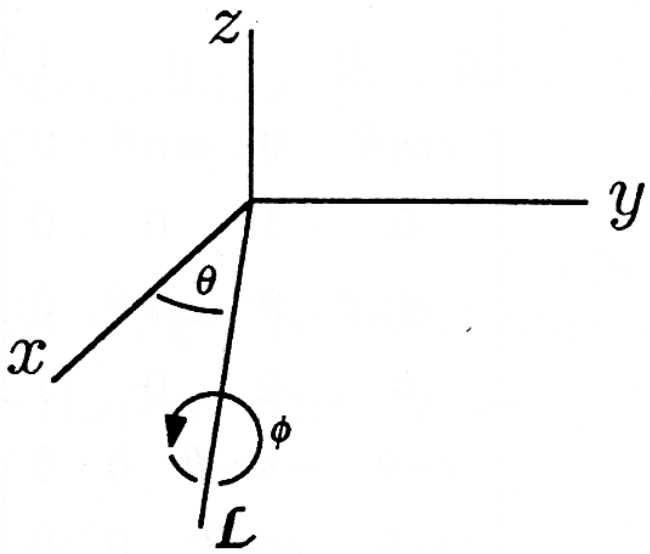
Directions of Positive Rotation

Example:
To rotate by angle φ about the line $\mathscr{L}$, which lies in the $x-y$ plane in Figure 3, we would

i. rotate $\mathscr{L}$ to the x-axis with $\mathrm{R}_z(-\theta)$;
ii. rotate by φ about the x-axis with $\mathrm{R}_x(\varphi)$; and
iii. rotate back to $\mathscr{L}$ with $\mathrm{R}_z(\theta)$.

The composite operation would be
Equation:

$$\mathrm{A}(\theta,\varphi)=\mathrm{R}_z(\theta)\mathrm{R}_x(\varphi)\mathrm{R}_z(-\theta).$$



Composition of Rotations

Direction Cosines. As discussed in the chapter on Linear Algebra, a vector v may be specified by its coordinates (x, y, z) or by its length and direction. The length of v is $||\mathrm{v}||$, and the direction can be specified in terms of the three *direction cosines* ($\cos\,\theta_x$, $\cos\,\theta_y$, $\cos\,\theta_z$). The angle θ_x is measured between the vector v and the x-axis or, equivalently, between the vector v and the vector $\mathrm{e}_x = [100]^T$. We have

Equation:

$$\cos\,\theta_x = \frac{(\mathrm{v}, \mathrm{e}_x)}{||\mathrm{v}||||\mathrm{e}_x\,||} = \frac{x}{||\mathrm{v}||}$$

Likewise, θ_y is measured between v and $\mathrm{e}_y = [010]^T$, and θ_z is measured between v and $\mathrm{e}_z = [001]^T$. Thus

Equation:

$$\cos\,\theta_y = \frac{y}{||\mathrm{v}||}$$
$$\cos\,\theta_z = \frac{z}{||\mathrm{v}||}$$

The vector
Equation:

$$\mathrm{u} = \begin{matrix} \cos\,\theta_x \\ \cos\,\theta_y \\ \cos\,\theta_z \end{matrix}$$

is a unit vector in the direction of v, so we have
Equation:

$$\mathrm{v} = ||\mathrm{v}||\mathrm{u} = ||\mathrm{v}|| \begin{matrix} \cos\,\theta_x \\ \cos\,\theta_y \\ \cos\,\theta_z \end{matrix} .$$

Exercise:

Problem: Show that u is a unit vector (i.e. 1 u$||$ = 1).

The direction cosines are useful for specifying a line in space. Instead of giving two points P_1 and P_2 on the line, we can give one point P_1 plus the direction cosines of any vector that points along the line. One such vector is the vector from P_1 to P_2.

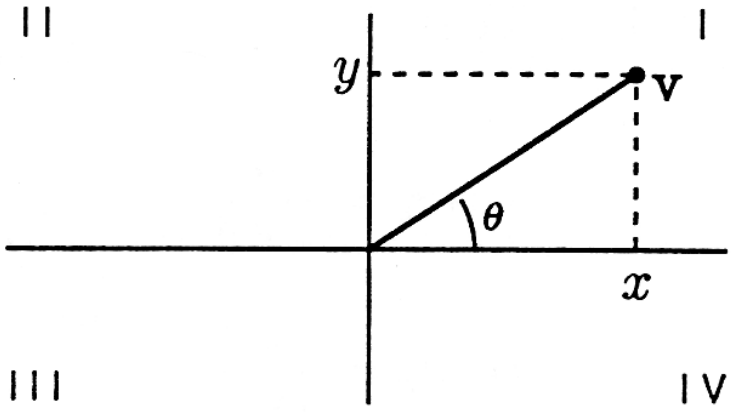
Arc tangents. Consider a vector $\mathrm{v} = \begin{Bmatrix} x \\ y \end{Bmatrix}$ in two dimensions. We know that
Equation:

$$\tan\,\theta = \frac{y}{x}$$

for the angle θ shown in Figure 4. If we know x and y, we can find θ using the arc tangent function

Equation:

$$\theta = \tan^{-1}\left(\frac{y}{x}\right).$$



Tangent and Arc Tangent

In MATLAB,

```
theta = atan(y/x)
```

Unfortunately, the arc tangent always gives answers between $\frac{-\pi}{2}$ and $\frac{\pi}{2}$ corresponding to points v in quadrants I and IV. The problem is that the ratio $\frac{-y}{-x}$ is the same as the ratio $--$ $-x$A so quadrant III cannot be distinguished from quadrant I by the ratio $\frac{y}{x}$ Likewise, quadrants II and IV are indistinguishable.

The solution is the two-argument arc tangent function. In MATLAB,

```
theta = atan2(y,x)
```

will give the true angle θ $-\pi$ and π in any of the four quadrants.

Example:

Consider the direction vector $\mathrm{u} = \begin{matrix} \cos\theta_x \\ \cos\theta_y \\ \cos\theta_z \end{matrix}$, as shown in Figure 5. What is the angle φ_y between the projection of u into the $x - y$ plane and the y-axis? This is important because it is $\mathrm{R}_z\left(\varphi_y\right)$ that will put u in the $y - z$ plane, and we need to know the angle φ_y in order to carry out this rotation. First note that it is *not* the same as θ_y. From the geometry of the figure, we can write

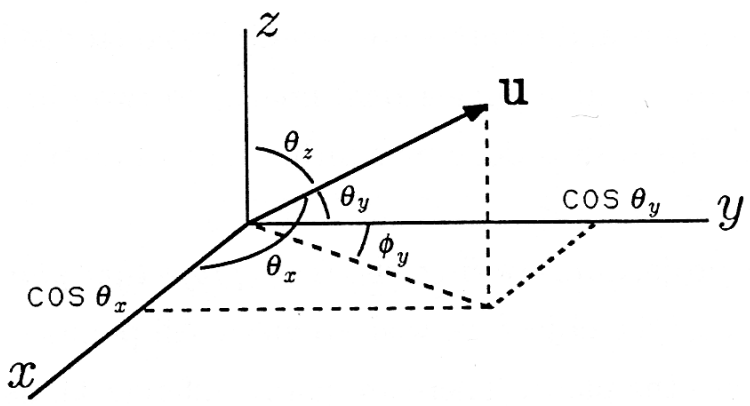
Equation:

$$\tan\,\varphi_y = \frac{\cos\,\theta_x}{\cos\,\theta_y}$$

This gives us a formula for φ_y in terms of the direction cosines of u. With the two-argument arc tangent, we can write

Equation:

$$\varphi_y = \tan^{-1}\left(\cos\,\theta_x, \cos\,\theta_y\right).$$



Angles in Three Dimensions

Exercise:

Problem:

a. Suppose point $p\prime$ is in the $y-z$ plane in three dimensions, $\mathrm{p}'=(0, y','', 1)$. Find θ so that $\mathrm{R}_x(\theta)$ will rotate $p\prime$ to the positive z-axis. (Hint: Use the two-argument arc tangent. θ will be a function of $y\prime$ and z'.)

b. Let p be any point in three-dimensional space, $\mathrm{p}=(x, y, z, 1)$. Find ψ so that $\mathrm{R}_z(\psi)$ rotates p into the $y-z$ plane. (Hint: Sketch the situation in three-dimensions, then sketch a two-dimensional view looking down at the $x-y$ plane from the positive z-axis. Compare with Example 2.) Your answers to parts (a) and (b) can be composed into an operator $\mathrm{Z}(\mathrm{p})$ that rotates a given point p to the positive z-axis, $\mathrm{Z}(\mathrm{p})=\mathrm{R}_x(\theta)\mathrm{R}_z(\psi)$.

c. Let $\mathscr{L}$ be a line in three-dimensional space specified by a point 1 $=(x, y, z, 1)$ and the direction cosines (cos θ_x, cos θ_y, cos θ_z). Use the following procedure to derive a composite operator $\mathrm{R}(\varphi, \mathscr{L})$ that rotates by angle φ about the line $\mathscr{L}$:

   i. translate 1 to the origin;
   ii. let u =(cos θ_x, cos θ_y, cos θ_z, 1) and use $\mathrm{Z}(\mathrm{u})$ to align $\mathscr{L}$ with the z-axis;

iii. rotate by φ about the z-axis;
iv. undo step (ii); and
v. undo step (i).

Vector Graphics: Projections

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

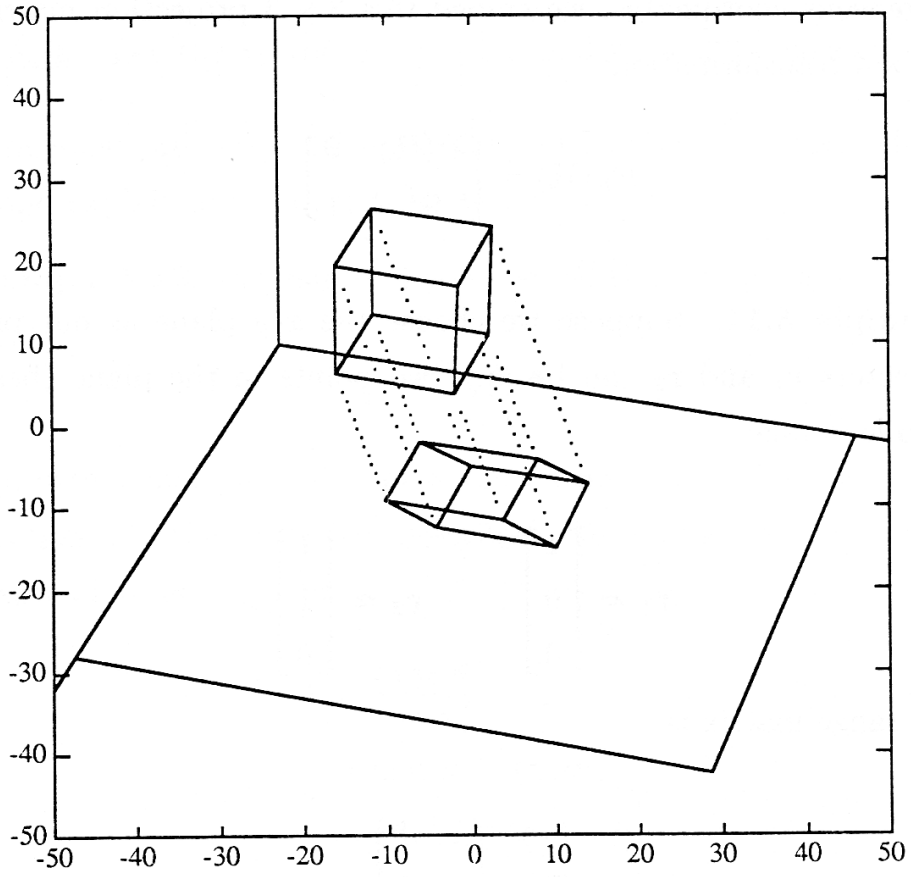
Computer screens and printers are two-dimensional display devices. We must somehow convert three-dimensional images to two dimensions in order to display them. This task is done by another kind of matrix operator called *a projection*.

To build a projection, we first choose a *projection plane* in the three-dimensional space of the object we wish to view. All points of the object are then projected onto the plane. There are many different kinds of projections, corresponding to various geometric rules for mapping points in space onto a plane. We begin with the *parallel projection* illustrated in Figure 1, wherein the dotted lines between the points and their projections in the plane are all parallel to one another. These dotted lines are called *projectors*.

Range. The projection plane is called the *range* of the projection. We will assume for now that the projection plane passes through the origin. If it does not, we may later compose the necessary translations with our projection. Three points, not all in a line, are required to determine a plane. We will take the origin as one of the points and suppose that the other two points are the vectors r_1 and r_2. From these vectors, we form the matrix
Equation:

$$\mathrm{R} = [\mathrm{r}_1 \mathrm{r}_2],$$

which determines the range of the projection.



Oblique Parallel Projection

Null-Space. To complete the specification of the parallel projection, we must specify the direction of the projectors. The line through the origin in this direction is called the *null-space* of the projection. Note that any line in the three-dimensional object that is parallel to the null-space will disappear to a point when projected. The null-space may be specified by a vector n pointing in the direction of the projectors.

Orthogonal Projection. If the null-space is perpendicular to the range, we say that the projection is an *orthogonal projection*. The matrix operator for orthogonal projection is a function of the range. If we assume that R is specified in three-dimensional coordinates (not homogeneous!), we have the following definition for the orthogonal projection onto the range of R:

Equation:

$$\mathrm{P}\left(\mathrm{R}\right) = \mathrm{R}\left(\mathrm{R}^{T}\mathrm{R}\right)^{-1}\mathrm{R}^{T} \in \mathscr{R}^{3\times 3}.$$

In homogeneous coordinates, we embed this 3×3 projection matrix in the general 4×4 transformation:
Equation:

$$\mathrm{P}_{h}\left(\mathrm{R}\right) = \begin{bmatrix} \mathrm{P}(\mathrm{R}) & 0 \\ 0^{T} & 1 \end{bmatrix}.$$

Example:
Suppose we choose the $x - y$ plane as our projection plane. Vectors r_1 and r_2 can be any two points in the plane (besides the origin), so let's take
Equation:

$$\mathrm{r}_1 = \begin{matrix} 1 \\ 0 \\ 0 \end{matrix}, \mathrm{r}_2 = \begin{matrix} 1 \\ 1 \\ 0 \end{matrix}$$

Then the range matrix is
Equation:

$$\mathrm{R} = \begin{matrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{matrix}.$$

The orthogonal projection looking down on the $x - y$ plane from the z-axis is
Equation:

$$\begin{aligned}\mathrm{P}(\mathrm{R}) &= \begin{matrix}1 & 1\\ 0 & 1\\ 0 & 0\end{matrix} \begin{bmatrix}1 & 0 & 0\\ 1 & 1 & 0\end{bmatrix} \begin{matrix}1 & 1\\ 0 & 1\\ 0 & 0\end{matrix}^{-1} \begin{bmatrix}1 & 0 & 0\\ 1 & 1 & 0\end{bmatrix} \\ &= \begin{matrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{matrix}.\end{aligned}$$

In homogeneous coordinates,
Equation:

$$\mathrm{P}_h(\mathrm{R}) = \begin{matrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\end{matrix}.$$

Exercise:

Problem:

Let the vectors $\mathrm{r}_1 = \begin{matrix}2\\0\\0\end{matrix}$ and $\mathrm{r}_2 = \begin{matrix}0\\3\\0\end{matrix}$ specify the range of an orthogonal projection. Find $\mathrm{P}(\mathrm{R})$ and $\mathrm{P}_h(\mathrm{R})$. Compare with Example 1 and explain any similarities.

Example:
In the chapter on linear algebra you learned that the projection of w onto x is given by
Equation:

$$\mathbf{z} = \frac{(\mathbf{x}, \mathbf{w})\mathbf{x}}{(\mathbf{x}, \mathbf{x})}$$

This is an orthogonal projection of *w* onto a line, but it is closely related to the projection into the plane just described. To see the similarity, let's work on the expression for z a little:
Equation:

$$\begin{aligned} \mathbf{z} &= \frac{\mathbf{x}(\mathbf{x},\mathbf{w})}{(\mathbf{x},\mathbf{x})} \\ &= \mathbf{x}(\mathbf{x}, \mathbf{x})^{-1} (\mathbf{x}, \mathbf{w}) \\ &= \mathbf{x}\left(\mathbf{x}^T\mathbf{x}\right)^{-1} \left(\mathbf{x}^T\mathbf{w}\right) \\ &= \left[\mathbf{x}\left(\mathbf{x}^T\mathbf{x}\right)^{-1}\mathbf{x}^T\right]\mathbf{w} \\ &= \mathrm{P}(\mathbf{x})\mathbf{w}. \end{aligned}$$

So we see that it can be written in the same way as the projection onto the plane. The only difference is that the range is now one-dimensional and specified by x in place of R.

Oblique Projection. More generally, we may have a null-space n that is not perpendicular to the range R. The projection shown in Figure 1 is an oblique projection. Once again we start with nonhomogeneous coordinates for n and R and write the 3×3 oblique projection as
Equation:

$$\mathrm{E}\left(\mathrm{R}, \mathrm{n}\right) = \mathrm{R}\left\{\mathrm{R}^T\left[I - -P(n)\right]\mathrm{R}\right\}^{-1}\mathrm{R}^T\left[I - -P(n)\right]$$

where
Equation:

$$\mathrm{P}\left(\mathrm{n}\right) = \mathrm{n}\left(\mathrm{n}^T\mathrm{n}\right)^{-1}\mathrm{n}^T = \frac{\mathrm{n}\mathrm{n}^T}{\left|\left|\mathrm{n}\right|\right|^2}$$

As with the orthogonal projection, we can return to homogeneous coordinates by

Equation:

$$\mathrm{E}_h\left(\mathrm{R}, \mathrm{n}\right) = \begin{bmatrix} \mathrm{E}(\mathrm{R} & \mathrm{n}) & 0 \\ \mathrm{O}^T & & 1 \end{bmatrix}.$$

Exercise:

Problem:

Prove and interpret the following properties of paralle1 projections (both orthogonal and oblique):

a. $\mathrm{P}^2 = \mathrm{P}; \mathrm{E}^2 = \mathrm{E}$.
b. $PR = \mathrm{R}; ER = \mathrm{R}$.
c. $En = 0; Pn = 0$. (First show that $\mathrm{R}^T \mathrm{n} = 0$ when *n* is orthogonal to R.)

Display from Projections. Even after we have used a projection, the image points are in three-dimensional homogeneous coordinates. How then do we display them on a two-dimensional display? If we had chosen the $x - y$ plane as the range of our projection, we could let the output device represent the $x - y$ plane and ignore the z-coordinate of each point. The z-coordinates of the projected image would all be 0 anyway since the projected points would lie in the $x - y$ plane. The fourth coordinate of 1 may also be ignored for display. But even if the projection plane is not the $x - y$ plane, we may perform a rotation to align the two planes and again let the output device represent the $x - y$ plane.

Perspective Projections. To obtain a perspective projection, we first choose a projection plane as we did for parallel projections. Instead of choosing a null-space parallel to all projectors, we now choose a viewpoint. The projectors all pass through the viewpoint in a perspective projection, as shown in Figure 2. For a viewpoint at $z = d$ on the z-axis and a projection plane coinciding with the $x - y$ plane, the three-dimensional homogeneous operator for perspective projection is

Equation:

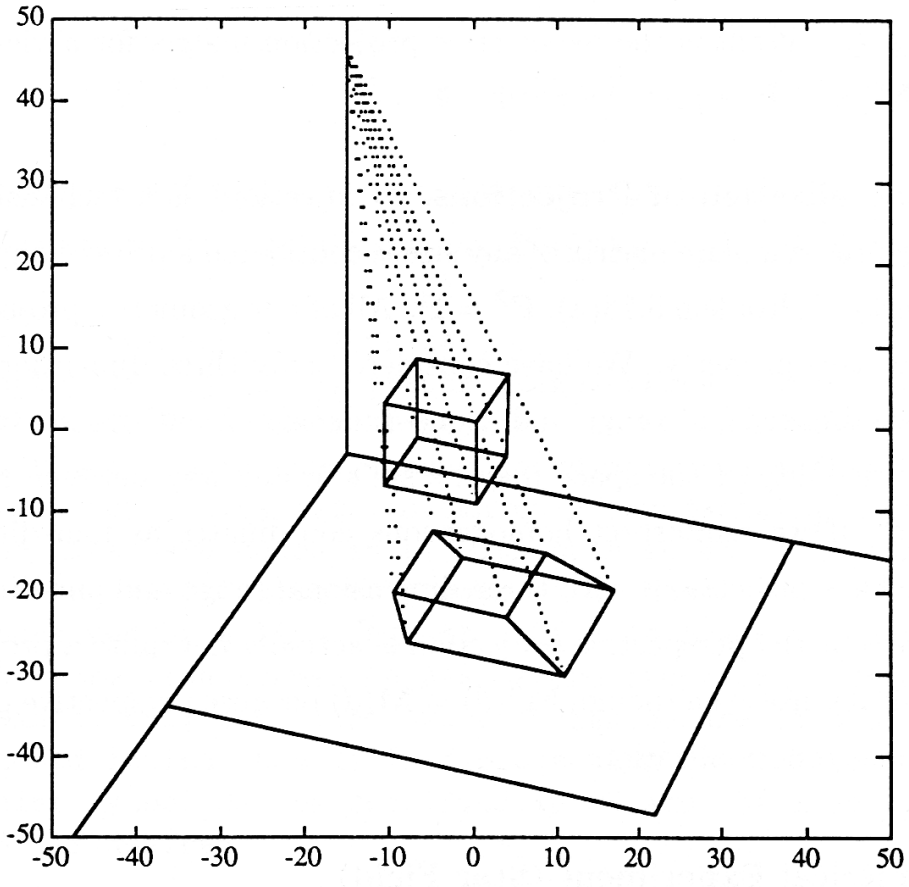
$$\mathrm{M}\left(d\right) = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{-1}{d} & 1 \end{matrix}.$$

The first thing you should notice about the perspective projection $\mathrm{M}(d)$ is that it violates the structure given in Equation 2 from "Vector Graphics: Three-Dimensional Homogeneous Coordinates" by having the 4,3 position equal to $-\frac{1}{d}$ rather than to 0. This means that the fourth coordinates will not remain 1 in the new point matrix

Equation:

$$\mathrm{G}_{\mathrm{new}} = \mathrm{M}\left(d\right)\mathrm{G}.$$

$$\mathbf{G_{new}} = \mathbf{M}(d)\mathbf{G}.$$



A Perspective Projection

This situation is interpreted to mean that the new point (x, y, z, w) must be *renormalized* to $\left(\frac{x}{w}, \frac{\mathrm{y}}{w} \frac{z}{w} 1\right)$ before the operation is considered complete. Renormalization adds another computational step to the image transformation process. This is undesirable, but we are stuck with it if we wish to include perspective projections in our repertoire of transformations. Note that renormalization is a point-by-point process since w may be different for each point.

How does renormalization affect composition of operators? We might expect that, each time we perform an operation requiring renormalization, we will be forced to stop and do the renormalization before going on with other operations. In this respect we are fortunate: we may put off the renormalization until all transformations are complete and renormalize once just before displaying the image. Thus we are able to compose perspective projections at will with other transformations.

Exercise:

Problem:

What is the perspective projection matrix for a viewpoint at infinity, $\mathrm{M}(\infty)$? Interpret the result.

Generalization of Projections. "Projection" is a technical term in linear algebra. A square matrix of any size is considered a projection if it obeys the property of Exercise 2(a), $\mathrm{P}^2 = \mathrm{P}$. All of our geometric projection matrices have this property. We have considered only three-dimensional spaces with a two-dimensional range and a one-dimensional null-space. In general, the dimensionality of the space may be split in any way between range and null-space. If we reinterpret homogeneous coordinates as four-dimensional space, we have projections with three-dimensional range and one-dimensional null-space, but the perspective projection is actually not quite a projection in the technical sense even though $\mathrm{M}^2\,(d) = \mathrm{M}\,(d)$ because perspective projection also includes renormalization.

Vector Graphics: Numerical Experiment (Star Field)

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

With Earth at the origin, we may specify the star positions for the Big and Little Dippers in three-dimensional homogeneous coordinates. With light years as our units, we have $\mathrm{G}^T =$

```
1.5441      -1.2064     153.0875    1.0000
-1.0386     8.4588      142.7458    1.0000
-8.7955     26.2870     198.0698    1.0000
-12.8807    19.0964     106.5383    1.0000
-18.8926    17.461      90.6185     1.0000
-45.1364    54.2706     215.1148    1.0000
-9.6222     20.1734     88.0062     1.0000
-33.7097    -8.4048     64.6574     1.0000
-33.7144    -8.7191     52.3806     1.0000
-43.8531    -1.1483     59.7193     1.0000
-36.1086    2.3667      55.7927     1.0000
-34.2309    8.2181      52.1260     1.0000
-30.7876    11.8182     46.9810     1.0000
-61.8581    31.5183     80.7615     1.0000
```

To make use of this data, we need a function to display it. Enter and save the following generalization of the function from Demo 1 in "Vector Graphics: Two Dimensional Image Representation". Call it `vhgraph.m`.

```
function vhgraph (P,L,A,ps,ls);
% vhgraph(P,L,A,PS,LS) graphs images whose points
are
```

```
% stored in P and whose lines are stored in L. The
 points
% in P must be in homogeneous coordinates in eithe
r 2 or 3
% dimensions, with each column of P representing a
 point.
% The lines are coded in L, with each column of L
containing
% 2 integers pointing to a pair of points in P to
be
% connected by a line segment. If A is present, th
e points
% in P are transformed to A*P beforre graphing. Fo
r 3D data
% points, only the first two coordinates are graph
ed, so A
% should include the desired projection from 3D to
 2D. The
% point symbol may be specified in PS and the line
 type in
% LS, if desired
%
% Richard T. Behrens, October 1989.
%
% The first dection of the program determines the
sizes of
% all the input matrices and chacks that they make
 sense.
[mp,np] = size(P);
if (nargin > 1)
   [ml,nl] = size(L);
else
    ml = 2; nl = 0;
end
if (nargin <=2)
    A = eye(mp);
end
```

```
[mA,nA] = size(A);
if ((mp~= nA) | (ml~=2))
    error('Incompatible sizes of input matrices.")
end
if (nargin <= 3);
    ps = '*';
end

P = A*P;    % Performs the transformation A on the
 points
            % (effect is only local to this functi
on).

% The next section contains a loop that renormaliz
es the
% homogeneous coordinates of the points if necessa
ry.
renorm = find((P(mA,:)~= 1));
if~isempty(renorm)
    for i = 1:length(renorm)
        P(:,renorm(i)) = P(:,renorm(i))/P(mA,renor
m(i));
    end
end

% The next program line sets a fixed scale output
window
% from -50 to 50 in both x and y directions on the
 screen.
% For automatic scaling to include all points of t
he
% image, we could use instead the line q = min(min
(P));
% r = max(max(P));
q = -50; r = 50;
axis([q r q r])
axis('square')
```

```
plot(P(1,:),P(2,:),ps)  % Plots the points with sy
mbol ps.
hold on                 % Saves the points while w
e plot
for i = 1:nl            % lines with line type LS.
plot([P(1,L(1,i))P(1,L(2,i))],..
     [P(2,L(1,i))P(2,L(2,i))],ls)
end
hold off
```

Enter the point matrix given at rhe beginning of this section (and take its transpose to put it in the usual form). Also enter the line matrix from Demo 1 in "Vector Graphics: Two Dimensional Image Representation". Save these two matrices and try looking at the image

```
≫ save dip3d
≫ vhgraph(G,H)
```

No dippers in sight, right? Without specifying a transformation matrix A, we have defaulted to looking down on the $x-y$ plane from $z=\infty$ (a parallel projection). This is how the constellations would look from a distant galaxy (say, a billion light years north of here) through an enormous telescope. We need a perspective view from the origin (Earth), but first we need a set of functions to give us the fundamental operators with which we can build the desired projection.

Take $\mathrm{R}_y(\theta)$ as an example. The function to build it looks like

```
    function Ry = vhry (theta) ;
% Rotation matrix about y-axis for 3-D homogeneous
% coordinates.
    Ry = eye(4);
    Ry(1,1) = cos(theta);
    Ry(3,3) = cos(theta);
    Ry(3,1) = -sin(theta);
    Ry(1,3) = sin(theta);
```

Enter and save `hry.m` as given. Then write functions for

Equation:

$$\mathrm{R}_x(\theta)\ vhrx.\ \mathrm{m}$$
$$\mathrm{R}_z(\theta)\ vhrz.\ \mathrm{m}$$
$$\mathrm{S}(s_x, s_y, s_z)\ vhs.\ \mathrm{m}$$
$$\mathrm{T}(t_x, t_y, t_z)\ vht.\ \mathrm{m}$$
$$\mathrm{M}(d)\ vhm.\ \mathrm{m}$$

Useful MATLAB functions for this task include `zeros`, `eye`, and `diag`.

Now build and use a perspective projection with viewpoint Earth and projection plane at $z = 1000$ behind the dippers:

1. translate Earth to $z = -1000$ so that the projection plane coincides with the $x - y$ plane: $\mathrm{T}(\mathrm{O}, 0, -1000)$;
2. use the fundamental perspective projection: $\mathrm{M}(-1000)$; and
3. translate back: $\mathrm{T}(\mathrm{O}, 0, 1000)$.

```
≫ A = vht(0,0,1000) * vhm(-1000) * vht(0,0,-1000)
≫ vhgraph(G,H,A)
```

Oops! Now the image is too big; it's mostly off the screen. Scale it down and have another look:

```
≫ A = vhs(.06,.06,.06) * A
≫ vhgraph(G,H,A)
```

Now the view should look familiar. Leave *A* as it is now:

```
≫ save dip3d
```

Experiment with scale and rotation about the z-axis. For example, try

```
≫ vhgraph(G,H,vhrz(pi/2) * A)
```

The two-dimensional star positions given in Demo 1 in "Vector Graphics: Two Dimensional Image Representation" were obtained from the three-dimensional positions with the composite operator A you are now using. To compare the two, type

```
≫ Gnew = A * G

≫ for i = 1:14
Gnew(;,i) = Gnew(:,i)/Gnew(4,i);
end

≫ Gnew
```

and compare the x and y coordinates with those of Demo 1 in "Vector Graphics: Two Dimensional Image Representation".

Astronomers give star positions in equatorial coordinates using right ascension, declination, and distance. The following function converts equatorial coordinates, which are spherical, to Cartesian coordinates with the z-axis pointing north, the x-axis pointing at the vernal (Spring) equinox in the constellation Pisces, and the y-axis pointing toward the Winter solstice in the constellation Opheuchus.

```
function v = starxyz(rah,ram,decd,decm,dist)
% starxyz is the cartesian coordinates of a star w
hose
% spherical coordinates ( e.g. from a star catalog
) are
%
%          rah     right ascension hours
%          ram     right ascension minutes
%          decd    declination degrees
%          decm    declination minutes (should be
negative
%                  if decd is negative)
%          dist    distance (light years)
%
```

```
phi = (pi/180) * (decd + decm/60);
theta = (pi/12) * (rah + ram/60);
r = dist;
v = [r * cos(phi) * cos(theta); -
r * cos(phi) * sin(theta);
    r * sin(phi)];
```

Have you ever wondered what the constellations would look like from other places in the galaxy? We will soon see the answer. First we will view the dippers from Alpha Centauri, the nearest star, whose coordinates are

```
 -1.5680     1.3157      -3.6675.
```

We will look toward the *centroid* of the fourteen stars in the dippers, located at

```
-26.3632      12.8709      100.4714.
```

To get the desired view, we must

1. translate the viewpoint to the origin;
2. rotate the centroid (direction to look) to the z-axis–note that the centroid will have new coordinates after step (1); and
3. apply the composite `A=S(.06,.06,.06)*T(0,0,1000)*M(-1000)*T(0,0,-1000)` (as used to view from Earth).

Write a function `vhz.m` based on Exercise 5 from "Vector Graphics: Three-Dimensional Homogeneous Coordinates" to accomplish step (2). Test it on several random points to make sure it works right. Now write a general perspective projection function called `vhview.m`. The function `vhview` should accept as inputs two vectors, the first specifying the viewpoint and the second the point to look toward. Its output should be a composite operator that performs all three of the preceding steps.

Now we want to look toward the centroid of the dippers from Alpha Centauri. To do so, enter the vectors for the two points of interest and construct the view like this:

```
≫ acentauri = [-1.5680; 1.3157; -3.6675]
≫ centroid = [-26.3632; 12.8709; 100.4714]
≫ A = vhview(acentauri,centroid)
≫ vhgraph(G,H,A)
≫ title('From Alpha Centauri')
```

The farther we move from Earth, the more distorted the dippers will look in general. It should be easy now to view them from any desired location. Just choose a viewpoint, recalculate the composite operator for that viewpoint using `vhview`, and use `vhgraph`. Follow this procedure to view the dippers from each of the stars in the following list. You will need to use `starxyz` first to convert their coordinates.

Star	Right Ascension	Declination	Distance (ly)
Alpha Centauri	14h 40m	-60° 50'	4.2
Sirius	6h 45m	-16°43'	9.5
Arcturus	14h 16m	19°11'	16.6
Pollux	7h 45m	28°02'	35.9
Betelgeuse	5h 55m	7°24'	313.5

Table 1 is a six-row and four-column table with the first row as a header, the first column describing different stars, and the values in the second through fourth columns describing the corresponding right ascension, declination, and distance for each star.

Of course, star viewing is not the only application of vector graphics. Do some experiments with the unit cube (see Exercise 2 from "Vector Graphics: Three-Dimensional Homogeneous Coordinates"). View the cube from location (4,3,2) looking toward the origin using the procedure just outlined for stars. You may need to adjust the scaling to get a meaningful view.

Filtering: Introduction

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Notes to Teachers and Students:

Filtering is one of the most important things that electrical and computer engineers do. In this chapter we extend everyday understanding of filters to *numerical* filters. We then study weighted moving averages and exponential averages. We define the important *test signals* for electrical and computer engineering and show how filters respond to them. The idea that filters are characterized by their response to simple test signals is fundamental. In the numerical experiment, students explore the *frequency response* of a simple filter, a concept that forms the basis of circuit theory, electronics, optics and lasers, solid-state devices, communications, and control.

Introduction

A *filter* is any device that passes material, light, sound, current, velocity, or information according to some rule of selectivity. Material (or mechanical) filters are commonplace in your everyday life:

i. coffee filters pass flavored water while filtering out coffee grounds;
ii. Goretex fibers pass small, warm perspiration droplets while filtering out large, cool droplets of rain or snow;
iii. fiberglass strands in a furnace filter pass warm air while filtering out particles of dirt and dust;
iv. a centrifuge retains material of low density while spinning out (or filtering out) material of high density; and
v. an electrostatic precipitator filters out dust and other effluents by attaching charge to them and using an electric field to move the charged particles to a high potential drain.

The first three of these examples selectively pass material according to size; the last two selectively pass material according to its mass density.

Typical filters for light are

i. UV filters on camera lenses and eyeglasses that pass light in the range of visible wavelengths while blocking light in the invisible (but damaging) ultraviolet range;
ii. polaroid lenses that pass light that is randomly polarized while blocking out glare that is linearly polarized;
iii. green fabrics that reflect green light and absorb other colors;
iv. red taillights that pass light in the long wavelength red range and reflect light in the short wavelength violet range (look at the inside of your taillights to see violet); and
v. glacial ice that absorbs all but the blue wavelengths so that it appears blue.

Exercise:

Problem: List as many examples of natural and man-made sound filters as you can.

Satellite Television. Among current filters, the tuner in a super-heterodyne receiver is, perhaps, the first example that comes to mind. But satellite TV filters are another fascinating example. A typical C-band satellite has twelve transponders (or repeaters), each of which transmits microwave radiation in a personalized 36 MHz band. (The abbreviation MHz stands for megahertz, or 10^6 Hz, or 10^6 cycles per second. Other common abbreviations are Hz for 1 Hz, kHz for 10^3 Hz, and GHz for 10^9 Hz.) Each transponder actually transmits two channels of information, one vertically polarized and one horizontally polarized. There is an 8 MHz guard band between each band, and the vertical and horizontal channels are offset by 20 MHz. The transmission scheme for the 24 channels is illustrated in Figure 1. The entire transmission band extends over 540 MHz, from 3.7×10^9 Hz to 4.24×10^9 Hz. The satellite receiver has two different microwave detectors, one for vertical and one for horizontal polarization, and a microwave tuner to tune into the microwave band of interest.

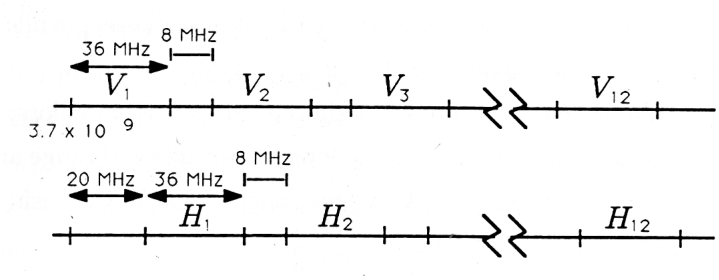
Exercise:

Problem: Check that the transmission scheme of Figure 1 consumes 540 MHz of bandwidth.

Exercise:

Problem:

List as many examples of natural and man-made devices for velocity filtering as you can.



Satellite TV– V_i, Vertically Polarized Channel i; H_j, Horizontally Polarized Channel j

An Aside on Hertz and Seconds. The abbreviation Hz stands for hertz, or cycles/second. It is used to describe the frequency of a sinusoidal signal. For example, house current is 60 Hz, meaning that it has 60 cycles each second. The inverse of Hz is seconds or, more precisely, seconds/cycle, the period of 1 cycle. For example, the period of 1 cycle for house current is 1/60 second. When we are dealing with sound, electricity, and electromagnetic radiation, we need a concise language for dealing with signals and waves whose frequencies range from 0 Hz (called DC or direct current) to 10^{18} Hz (visible light). Table 1 summarizes the terms and symbols used to describe the frequency and period of signals that range in frequency from 0 Hz to 10^{12} Hz.

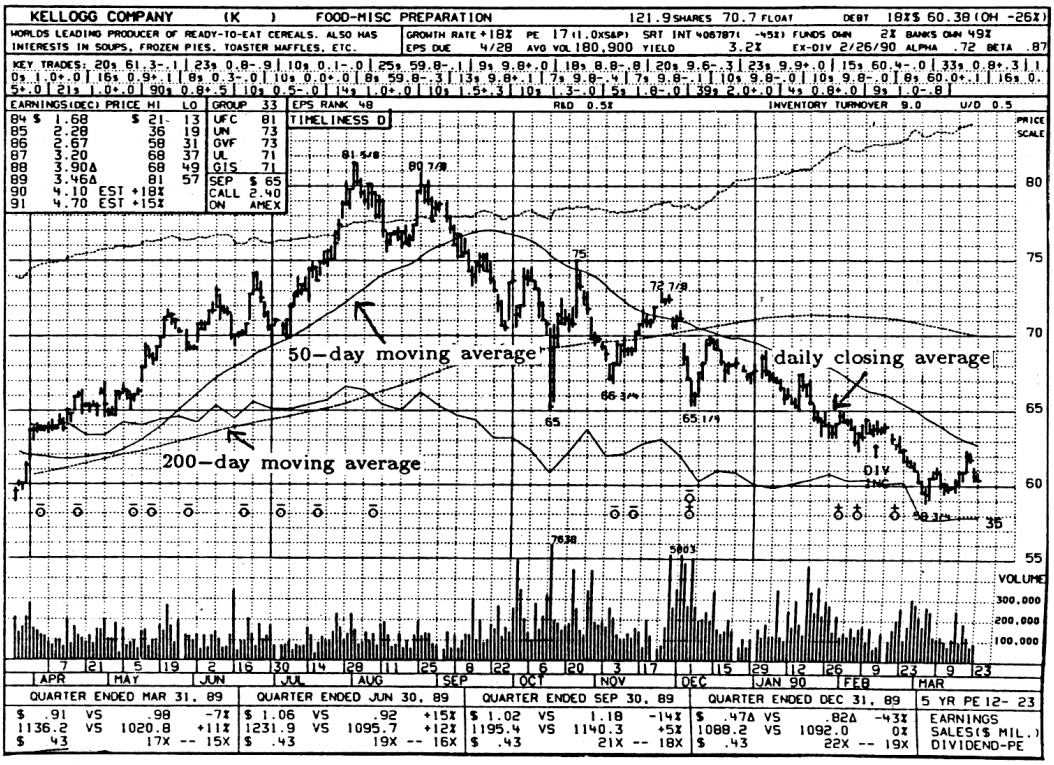
Frequency			Period			
Hz	**Term**	**Units**	**Seconds**	**Term**	**Units**	**Example**
Hz	hertz	1 Hz	sec	second	1 sec	battery current: 0 Hz house current: 60 Hz
kHz	kilohertz	10^3 Hz	msec	millisecond	10^{-3} sec	midfrequency sound
MHz	megahertz	10^6 Hz	μsec	microsecond	10^{-6}sec	clock frequencies in microcomputers
GHz	gigahertz	10^9 Hz	nsec	nanosecond	10^{-9}sec	microwave radiation for satellite communication
THz	terahertz	10^{12} Hz	psec	picosecond	10^{-12} sec	infrared radiation

Terms and symbols for Sinusoidal Signals

Numerical Filters. Rather amazingly, these ideas extend to the domain of numerical filters, the topic of this chapter. Numerical filters are just schemes for weighting and summing strings of numbers. Stock prices are typically averaged with numerical filters. The curves in Figure 2 illustrate the daily closing average for Kellogg's common stock and two moving averages. The 50-day moving average is obtained by passing the daily closing average through a numerical filter that averages the most current 50 days' worth of closing averages. The 200-day moving average for the stock price is obtained by passing the daily closing prices through a numerical filter that averages the most current 200 days' worth of daily closing averages. The daily closing averages show fine-grained variation but tend to conceal trends. The 50-day and 200-day averages show less fine-grained variation but give a clearer picture of trends. In fact, this is one of the key ideas in numerical filtering: by selecting our method of averaging, we can filter out fine-grained variations and pass long-term trends (or vice versa), or we can filter out periodic variations and pass nonperiodic variations (or vice versa). Figure 2 illustrates that moving averages typically lag increasing sequences of numbers and lead decreasing sequences. Can you explain why?

We will call any algorithm or procedure for transforming one set of numbers into another set of numbers a *numerical filter* or *digital filter*. Digital filters, consisting of memories and arithmetic logic units (ALUs), are implemented in VLSI circuits and used for communication, control, and instrumentation. They are also implemented in random–or semicustom–logic circuits and in programmable microcomputer systems. The inputs to a digital filter are typically electronic measurements that are produced by A/D (analog-to-digital) conversion of the output of an electrical or mechanical sensor. The outputs of the filter are “processed,” “filtered,” or “smoothed” versions of the measurements. In your more advanced courses in electrical and computer engineering you will study signal processing and system theory, assembly language programming, microprocessor system

development, and computer design. In these courses you will study the design and programming of hardware that may be used for digital filtering.



Dow-Jones Averages (Adapted from the New York Stock Exchange, *Daily Graphs*, William O'Neil and Co., Inc., Los Angeles, California)

Filtering: Simple Averages

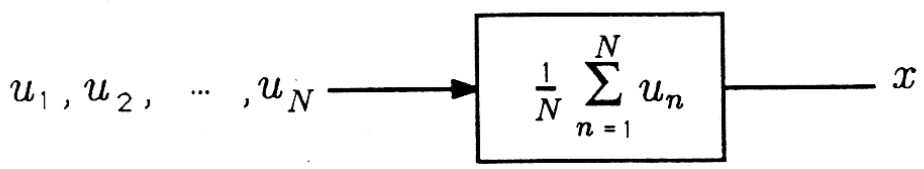
Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

The simplest numerical filter is the simple averaging filter. This filter is defined by the equation

Equation:

$$x = \frac{1}{N} \sum_{n=1}^{N} u_n.$$

The filter output *x* is the average of the *N* filter inputs $u_1, u_2, \ldots, u_N$. These inputs may be real or complex numbers, and *x* may be real or complex. This simple averaging filter is illustrated in Figure 1.



A Simple Averaging Filter

Example:

If the averaging filter is excited by the constant sequence $u_1 = u_2 = \cdots = u_N = u$, then the output is

Equation:

$$x = \frac{1}{N}\sum_{n=1}^{N} u = u.$$

The output is, truly, the average of the inputs. Now suppose the filter is excited by the linearly increasing sequence

Equation:

$$u_n = n, \;\; n = 1, 2, ..., N.$$

This sequence is plotted in Figure 2. How do we sum such a sequence in order to produce the average x? For N even, the average may be written as

Equation:

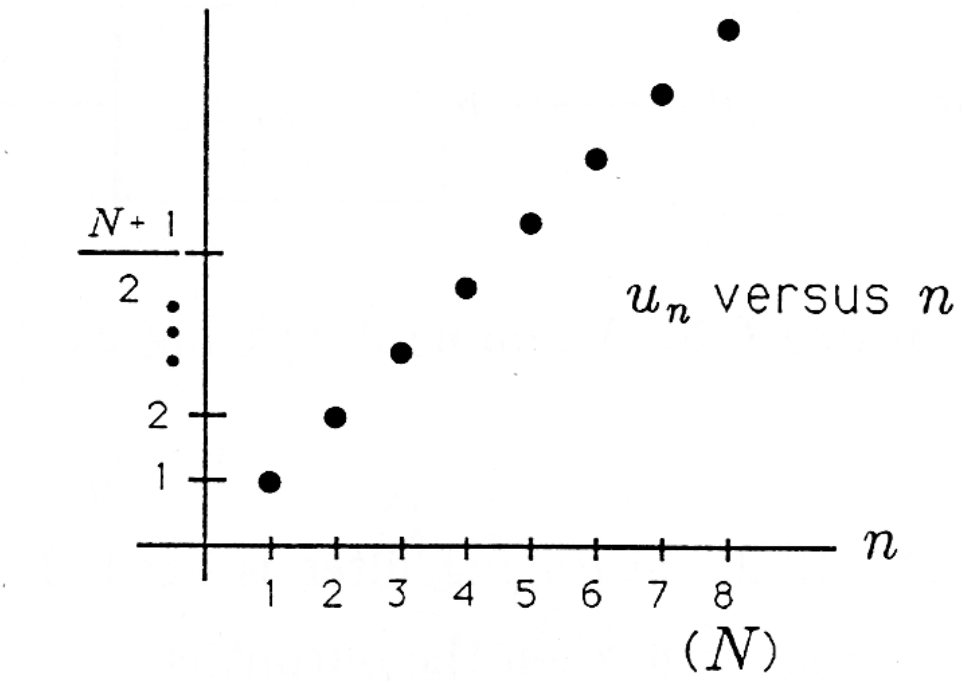
$$x = \frac{1}{N}(x_1 + x_N) + \frac{1}{N}(x_2 + x_{N-1}) + \cdots + \frac{1}{N}\left(x_{N/2} + x_{(N/2)+1}\right).$$

Each pair-sum in parentheses equals $N + 1$, and there are $\frac{N}{2}$ such pair-sums, so the average is

Equation:

$$x = \frac{1}{N}\frac{N}{2}(N + 1) = \frac{N + 1}{2}.$$

This is certainly a reasonable answer for the average of a linearly increasing sequence. See Figure 2.



Linearly Increasing Sequence

Exercise:

Problem:

Write $x = \frac{1}{N}\sum_{n=1}^{N} n$ as a sum of pair-sums for N odd. What does x equal?

General Sum Formula. Suppose the input to the simple averaging filter is the polynomial sequence
Equation:

$$u_n = n^k, \;\; n = 1, 2, ..., N$$

where k is a non-negative integer such as $k = 0, 1, 2, ...$ The output of the filter is
Equation:

$$x_N^{(k)} = \frac{1}{N} \sum_{n=1}^{N} n^k.$$

We rewrite x as $x_N^{(k)}$ to remind ourselves that we are averaging N numbers, each of which is n^k. For example, when $N = 8$ and $k = 2$,
Equation:

$$x_8^{(2)} = \frac{1}{8} \sum_{n=1}^{8} n^2 = \frac{1}{8}(1 + 4 + 9 + \cdots + 64).$$

Rather than study the average $x_N^{(k)}$, we will study the sum $Nx_N^{(k)}$ and divide by N at the very end:
Equation:

$$S_N^{(k)} = Nx_N^{(k)} = \sum_{n=1}^{N} n^k.$$

The sum $S_N^{(k)}$ may be rewritten as the sum
Equation:

$$\begin{aligned} S_N^{(k)} &= \textstyle\sum_{n=1}^{N-1} n^k + N^k \\ &= S_{N-1}^{(k)} + N^k. \end{aligned}$$

This result is very important because it tells us that the sum $S_N^{(k)}$, viewed as a function of N, obeys a *recursion* in which $S_N^{(k)}$ is just the sum using one

less input, namely, $S_{N-1}^{(k)}$, plus N^k. Now, since polynomials are the most general functions that obey such recursions, we know that $s_N^{(k)}$ must be a polynomial of order $k+1$ in the variable N:
Equation:

$$s_N^{(k)} = a_0 + a_1 N + a_2 N^2 + \cdots + a_{k+1} N^{k+1}.$$

Let's check to see that this polynomial really can obey the required recursion. First note that $S_{N-1}^{(k)}$ is the following polynomial:
Equation:

$$s_{N-1}^{(k)} = a_0 + a_1 \left(N-1\right) + \cdots + a_{k+1} (N-1)^{k+1}.$$

The term $(N-1)^{k+1}$ produces $\binom{k+1}{0} N^{k+1} (-1)^0 + \binom{k+1}{1} N^k (-1)^1 + \cdots$. (Remember the binomial expansion?) Therefore the difference between $S_N^{(k)}$ and $S_{N-1}^{(k)}$ is
Equation:

$$S_N^{(k)} - S_{N-1}^{(k)} = c_0 + c_1 N + \cdots + c_k N^k.$$

This recursion is general enough to produce the difference N^k provided we can solve for $a_0, a_1, \ldots, a_{k+1}$ to make $c_0 = c_1 = \cdots = c_{k-1} = 0$ and $c_k = 1$. We know that $S_N^{(k)} = 0$ for $N = 0$, so we know that $a_0 = 0$, meaning that the polynomial for $S_N^{(k)}$ can really be written as
Equation:

$$S_N^{(k)} = a_1 N + a_2 N^2 + \cdots + a_{k+1} N^{k+1}.$$

In order to solve for the coefficients of this polynomial, we propose to write out our equation for $s_N^{(k)}$ as follows:
Equation:

$$
\begin{array}{lll}
(N=1) & S_1^{(k)} = & a_1 + \cdots + a_{k+1} \\
(N=2) & S_2^{(k)} = & 2a_1 + \cdots + 2^{k+1}a_{k+1} \\
(N=3) & s_3^{(k)} = & 3a_1 + \cdots + 3^{k+1}a_{k+1} \\
\vdots & & \vdots \\
(N=k) & S_k^{(k)} = & ka_1 + \cdots + k^{k+1}a_{k+1} \\
(N=k+1) & S_{k+1}^{(k)} = & (k+1)a_1 + \cdots + (k+1)^{k+1}a_{k+1}.
\end{array}
$$

Using the linear algebra we learned earlier, we may write these equations as the matrix equation

Equation:

$$
\begin{bmatrix}
1 & 1 & \ldots & 1 \\
2 & 4 & \ldots & 2^{k+1} \\
\vdots & \vdots & & \vdots \\
k & k^2 & \ldots & k^{k+1} \\
(k+1) & (k+1)^2 & \ldots & (k+1)^{k+1}
\end{bmatrix}
\begin{bmatrix}
a_1 \\ a_2 \\ \vdots \\ \\ a_{k+1}
\end{bmatrix}
=
\begin{bmatrix}
s_1^{(k)} \\ S_2^{(k)} \\ \vdots \\ \\ S_{k+1}^{(k)}
\end{bmatrix}.
$$

The terms on the right-hand side of the equal sign are “initial conditions” that tell us how the sum $S_N^{(k)}$ begins for $N = 1, 2, \ldots, k+1$. These initial conditions must be computed directly. (For example, $S_2^{(k)} = 1^k + 2^k$.) Then the linear system of $(k+1)$ equations in $(k+1)$ unknowns may be solved for $a_1, a_2, \ldots, a_{k+1}$. The solution for S_N^k is then complete, and we may use it to solve for S_N^k for arbitrary N.

Example:
When $k = 2$, we have the following equation for the coefficients a_1, a_2, and a_3 in the polynomial $S_N^{(2)} = a_1 N + a_2 N^2 + a_3 N^3$:

Equation:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 3 & 9 & 27 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1^2 \\ 1^2 + 2^2 \\ 1^2 + 2^2 + 3^2 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 14 \end{bmatrix}.$$

Exercise:

Problem:

Solve for a_1, a_2, a_3 in the linear equation of Example 2. Show that $S_N^{(2)} = a_1 N + a_2 N^2 + a_3 N^3$ obeys the recursion $S_N^{(2)} - S_{N-1}^{(2)} = N^2$.

Exercise:

Problem:

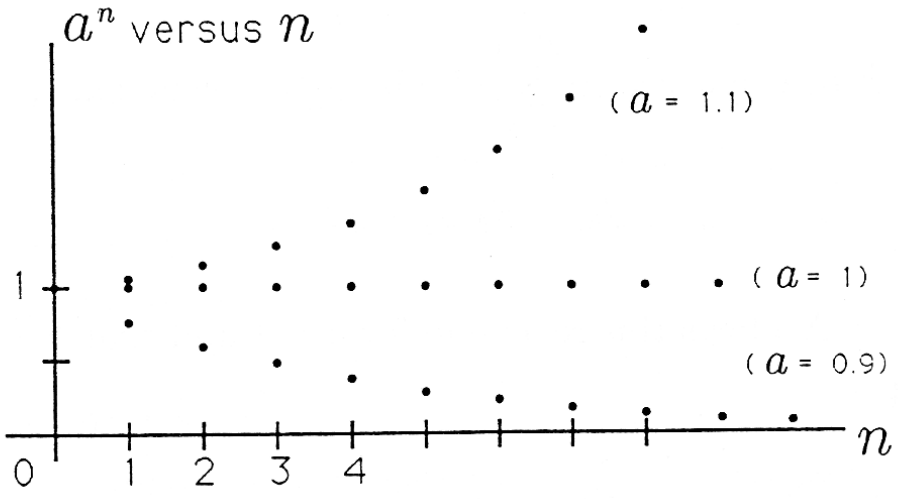
(MATLAB) Write a MATLAB program to determine the coefficients $a_1, a_2, ..., a_{k+1}$ for the polynomial $S_N^{(k)}$. Generate a table of formulas for the averages $x_N^{(k)}$ for $k = 1, 2, ..., 5$. Evaluate these formulas for $N = 2$, 4, 8, and 16.

Exponential Sums. When the input to an averaging filter is the sequence
Equation:

$$u_n = a^n, \;\; n = 0, 1, 2, ..., N - 1,$$

we say that the input is exponential (or geometric). Typical sequences are illustrated in Figure 6.5 for $a = 0.9$, $a = 1$, and $a = 1.1$. Don't let it throw you that we have changed the index to run from 0 to $N - 1$ rather than from 1 to N. This change is not fundamentally important, but it simplifies our study. The sum of the inputs is
Equation:

$$S_N = \sum_{n=0}^{N-1} a^n.$$



Exponential Sequences

How do we evaluate this sum? Well, we note that the sum aS_N is
Equation:

$$\begin{aligned} aS_N &= \sum_{n=0}^{N-1} a^{n+1} = \sum_{k=1}^{N} a^k \\ &= \sum_{k=0}^{N-1} a_k + a^N - 1 \\ &= S_N + a^N - 1. \end{aligned}$$

Therefore, provided $a \neq 1$, the sum S_N is
Equation:

$$S_N = \frac{1 - a^N}{1 - a}, \quad a \neq 1.$$

This formula, discovered already in the chapter covering the functions e^x and $e^{j\theta}$, works for $a \neq 1$. When $a = 1$, then $S_N = N$:

Equation:

$$S_N = \begin{cases} \frac{1-a^N}{1-a}, & a \neq 1 \\ N, & a = 1. \end{cases}$$

When $|a| < 1$, then $a^N \to 0$ for $N \to \infty$, and we have the asymptotic formula

Equation:

$$\lim_{N\to\infty} S_N = \frac{1}{1-a}, \quad |a| < 1.$$

Exercise:

Problem:

Evaluate $S_N = \sum_{n=0}^{N-1} a^n$ and $X_N = \frac{1}{N} S_N$ for $a = 0.9, 1$, and 1.1 and for $N = 1, 2, 4, 8, 16$, and 32.

Exercise:

Problem: Prove that $S_N = \sum_{n=0}^{N-1} a^n$ obeys the recursion

Equation:

$$S_N = S_{N-1} + a^{N-1}.$$

Prove that $S_N = N$ obeys this recursion for $a = 1$ and that $S_N = \frac{1-a^N}{1-a}$ obeys it for $a \neq 1$.

Recursive Computation. Every sum of the form
Equation:

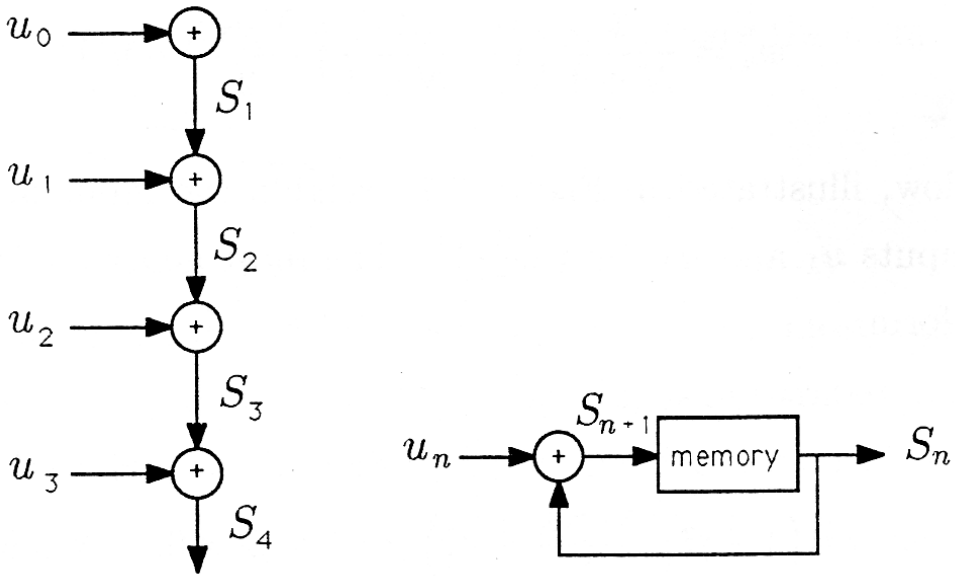
$$S_N = \sum_{n=0}^{N-1} u_n$$

obeys the recursion
Equation:

$$S_N = S_{N-1} + u_{N-1}.$$

This means that when summing numbers you may "use them and discard them." That is, you do not need to read them, store them, and sum them.

You may read u_0 to form S_1 and discard u_0; add u_1 to S_1 and discard u_1 ; add u_2 to S_2 ; and continue.



The Recursion $S_{n+1} = S_n + u_n$

This is very important for hardware and software implementations of running sums. You need only store the current sum, not the measurements that produced it. Two illustrations of the recursion $S_{n+1} = S_n + u_n$ are provided in Figure 4. The diagram on the left is self-explanatory. The diagram on the right says that the sum S_n is stored in a memory location, to be added to u_n to produce S_{n+1}, which is then stored back in the memory location to be added to u_{n+1}, and so on.

Filtering: Weighted Averages

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Weighted, tapered, or windowed averages are straightforward generalizations of simple averages. They take the form
Equation:

$$x = \sum_{n=1}^{N} w_n u_n$$

with the constraint that the "weights in the window," w_n, sum to 1:
Equation:

$$\sum_{n=1}^{N} w_n = 1.$$

When $w_n = \frac{1}{N}$ then x is the simple average studied in the section on "Simple Averages".

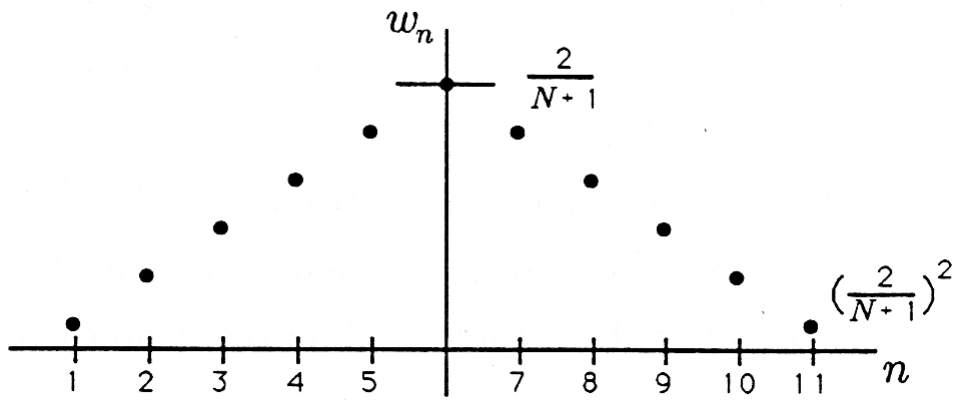
Example:
There are many windows that are commonly used in engineering practice. For N odd, the standard triangular window is
Equation:

$$w_n = \frac{2}{N+1}\left(1 - \frac{2}{N+1}\left|\frac{N+1}{2} - n\right|\right).$$

This window, illustrated in Figure 1, weights the input $u_{(N+1)/2}$ by $\frac{2}{N+1}$ and the inputs u_1 and u_N by $\left(\frac{2}{N+1}\right)^2$ The most general triangular window takes the form

Equation:

$$w_n = \alpha\left(1 - \beta\left|\frac{N+1}{2} - n\right|\right); \alpha, \beta > 0, \;\; N \;\text{ odd}.$$



Triangular Window

Exercise:

Problem:

Determine the constraints on α and β to make the general triangular window a valid window (i.e., $\sum_{n=1}^{N} w_n = 1$). Show that $\alpha = \frac{2}{N+1} = \beta$ is a valid solution. Propose another solution that you like.

Exercise:

Problem:

You are taking three 3-credit courses, one 5-credit course, and one 2-credit course. Write down the weighted average for computing your GPA in a system that awards 4.0 points for an A, 3.0 points for a B, ... , and (horrors!) 0 points for an F.

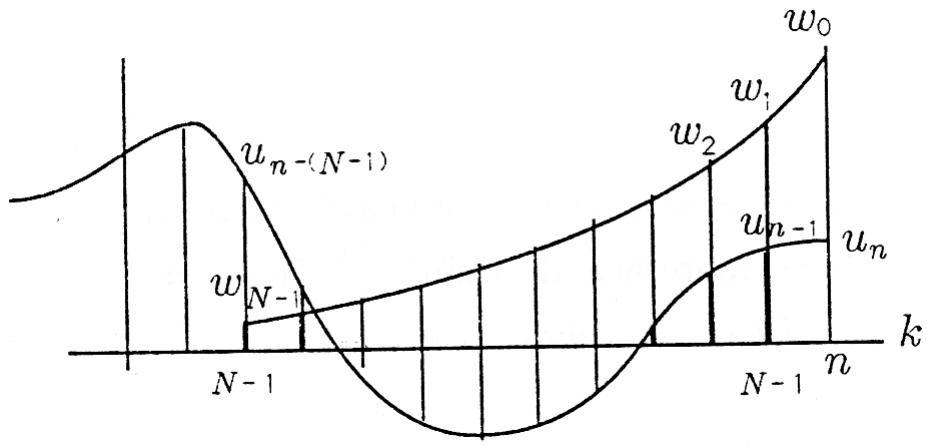
Filtering: Moving Averages

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Moving averages are generalizations of weighted averages. They are designed to "run along an input sequence, computing weighted averages as they go." A typical moving average over *N* inputs takes the form
Equation:

$$\begin{aligned} x_n &= \sum_{k=0}^{N-1} w_k u_{n-k} \\ &= w_0 u_n + w_1 u_{n-1} + \cdots + w_{N-1} u_{n-(N-1)}. \end{aligned}$$

The most current input, u_n, is weighted by w_0; the next most current input, u_{n-1}, is weighted by w_1 ; and so on. This weighting is illustrated in Figure 1. The sequence of weights, w_0 through w_{N-1}, is called a "window," a "weighting sequence," or a "filter." In the example illustrated in Figure 6.8, the current value u_n is weighted more heavily than the least current value. This is typical (but not essential) because we usually want x_n to reflect more of the recent past than the distant past.



Moving Average

Example:

When the weights $w_0, w_1, ..., w_{N-1}$ are all equal to $\frac{1}{N}$, then the moving average x_n is a "simple moving average":

Equation:

$$x_n = \frac{1}{N}[u_n + u_{n-1} + \cdots + u_{N-1}].$$

This is the same as the simple average, but now the simple average moves along the sequence of inputs, averaging the N most current values.

Exercise:

Problem:

Evaluate the moving average $x_n = \sum_{k=0}^{N-1} \frac{1}{N} u_{n-k}$ for the inputs

a. $u_n = \begin{cases} 0, & n < 0 \\ u, & n \geq 0; \end{cases}$

b. $u_n = \begin{cases} 0, & n \leq 0 \\ n, & n > 0. \end{cases}$

Interpret your findings.

Exercise:

Problem:

Evaluate the simple moving average $x_n = \sum_{k=0}^{N-1} \frac{1}{N} u_{n-k}$ when u_n is the sequence

Equation:

$$u_n = \begin{cases} 0, & n < 0 \\ a^n, & n \geq 0. \end{cases}$$

Interpret your result.

Example:
When the weights w_n equal $w_0 a^n$ for $n = 0, 1, ..., N-1$, then the moving average x_n takes the form

$$x_n = w_0 \sum_{k=0}^{N-1} a^k u_{n-k}.$$

When $a < 1$, then u_n is weighted more heavily than $u_{n-(N-1);}$ when $a > 1$, $u_{n-(N-1)}$ is weighted more heavily than u_n ; when $a = 1, u_n$ is weighted the same as $u_{n-(N-1)}$.

Exercise:

Problem:

Evaluate w_0 so that the exponential weighting sequence $w_n = w_0 a^n \ (n = 0, 1, ..., N-1)$ is a valid window (i.e., $\sum_{n=0}^{N-1} w_n = 1$).

Exercise:

Problem:

Compute the moving average $x_n = \sum_{k=0}^{N-1} w_0 a^k u_{n-k}$ when the input sequence u_n is

Equation:

$$u_n = \begin{cases} b^n, & n \geq 0 \\ 0, & n < 0. \end{cases}$$

What happens when $b = a$? Can you explain this?

Filtering: Exponential Averages and Recursive Filters

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Suppose we try to extend our method for computing finite moving averages to infinite moving averages of the form
Equation:

$$\begin{aligned} x_n &= \sum_{k=0}^{\infty} w_k u_{n-k} \\ &= w_0 u_n + w_1 u_{n-1} + \cdots + w_{1000} u_{n-1000} + \cdots \end{aligned}$$

In general, this moving average would require infinite memory for the weighting coefficients $w_0, w_1, ...$ and for the inputs $u_n, u_{n-1}, ...$ Furthermore, the hardware for multiplying $w_k u_{n-k}$ would have to be infinitely fast to compute the infinite moving average in finite time. All of this is clearly fanciful and implausible (not to mention impossible). But what if the weights take the exponential form
Equation:

$$w_k = \begin{cases} 0, & k < 0 \\ w_0 a^k, & k \geq 0? \end{cases}$$

Does any simplification result? There is hope because the weighting sequence obeys the recursion
Equation:

$$w_k = \begin{array}{ll} 0, & k < 0 \\ w_0, & k = 0 \\ aw_{k-1} & k \geq 1. \end{array}$$

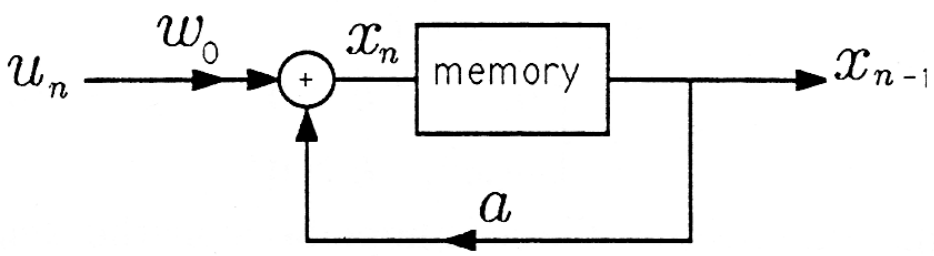
This recursion may be rewritten as follows, for $k \geq 1$:
Equation:

$$w_k - aw_{k-1} = 0, \;\; k \geq 1.$$

Let's now manipulate the infinite moving average and use the recursion for the weights to see what happens. You must follow every step:
Equation:

$$\begin{array}{rcc} x_n & = & \sum_{k=0}^{\infty} w_k u_{n-k} \\ & = & \sum_{k=1}^{\infty} w_k u_{n-k} + w_0 u_n \\ & = & \sum_{k=1}^{\infty} a w_{k-1} u_{n-k} + w_0 u_n \\ & = & a \sum_{m=0}^{\infty} w_m u_{n-1-m} + w_0 u_n \\ & = & a x_{n-1} + w_0 u_n. \end{array}$$

This result is fundamentally important because it says that the output of the *infinite exponential moving average* may be computed by scaling the previous output x_{n-1} by the constant a, scaling the new input u_n by w_0, and adding. Only three memory locations must be allocated: one for w_0, one for a, and one for x_{n-1}. Only two multiplies must be implemented: one for ax_{n-1} and one for $w_0 u_n$. A diagram of the recursion is given in Figure 1. In this recursion, the old value of the exponential moving average, x_{n-1}, is scaled by a and added to $w_0 u_n$ to produce the new exponential moving average x_n. This new value is stored in memory, where it becomes x_{n-1} in the next step of the recursion, and so on.



Recursive Implementation of an Exponential Moving Average

Exercise:

Problem:

Try to extend the recursion of the previous paragraphs to the weighted average

Equation:

$$x_n = \sum_{k=0}^{N-1} a^k u_{n-k}.$$

What goes wrong?

Exercise:

Problem:

Compute the output of the exponential moving average $x_n = ax_{n-1} + w_0 u_n$ when the input is

Equation:

$$u_n = \begin{cases} 0, & n < 0 \\ u, & n \geq 0. \end{cases}$$

Plot your result versus n.

Exercise:

Problem: Compute w_0 in the exponential weighting sequence
Equation:

$$w_n = \begin{cases} 0, & n < 0 \\ a^n w_0, & n \geq 0 \end{cases}$$

to make the weighting sequence a valid window. (This is a special case of Exercise 3 from Filtering: Moving Averages.) Assume $-1 < a < 1$

Filtering: Test Sequences

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

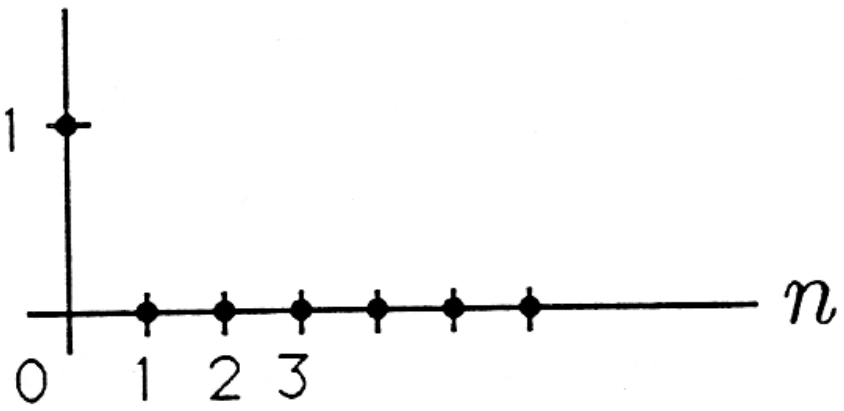
When we design a filter, we design it for a purpose. For example, a moving average filter is often designed to pass relatively constant data while averaging out relatively variable data. In an effort to clarify the behavior of a filter, we typically analyze its response to a standard set of test signals. We will call the *impulse*, the *step*, and the *complex exponential* the standard test signals.

Unit Pulse Sequence. The unit pulse sequence is the sequence
Equation:

$$u_n = \delta_n = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0. \end{cases}$$

This sequence, illustrated in Figure 1, consists of all zeros except for a single one at $n = 0$. If the unit pulse sequence is passed through a moving average filter (whether finite or not), then the output is called the *unit pulse response*:
Equation:

$$\begin{aligned} h_n &= \sum_{k=0}^{\infty} w_k \delta_{n-k} \\ &= w_n. \end{aligned}$$



Unit Pulse Sequence

(Note that $\delta_{n-k} = 0$ unless $n = k$.) So the unit pulse sequence may be used to read out the weights of a moving average filter. It is common practice to use w_k (the kth weight) and h_k (the kth impulse response) interchangeably.

Exercise:

Problem:

Find the unit pulse response for the finite moving average $x_n = \sum_{k=0}^{N-1} w_k u_{n-k}$. Caution: You must consider $n < 0, 0 \leq n \leq N-1$, and $n \geq N$.

Exercise:

Problem:

Find the unit pulse response for the recursive filter $x_n = ax_{n-1} + w_0 u_n$.

Unit Step Sequence. The unit step sequence is the sequence

Equation:

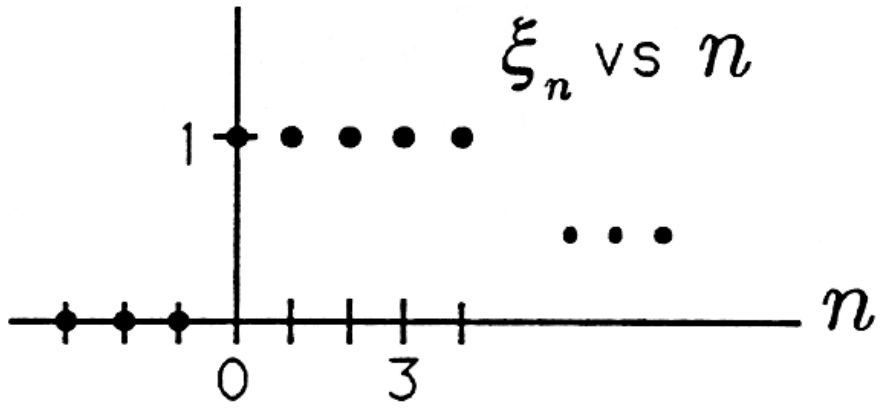
$$u_n = \xi_n = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0. \end{cases}$$

This sequence is illustrated in Figure 2. When this sequence is applied to a moving average filter, the result is the *unit step response*

Equation:

$$\begin{aligned} g_n &= \sum_{k=0}^{n} w_k \\ &= \sum_{k=0}^{n} h_k. \end{aligned}$$

The unit step response is just the sequence of partial sums of the unit pulse response.



Unit Step Sequence

Exercise:

Problem:

Find the unit step response for the finite moving average filter $x_n = \sum_{k=0}^{N-1} w_k u_{n-k}$. Specialize your general result to the special case where $w_k = \frac{1}{N}$ for $k = 0, 1, ..., N-1$.

Exercise:

Problem:

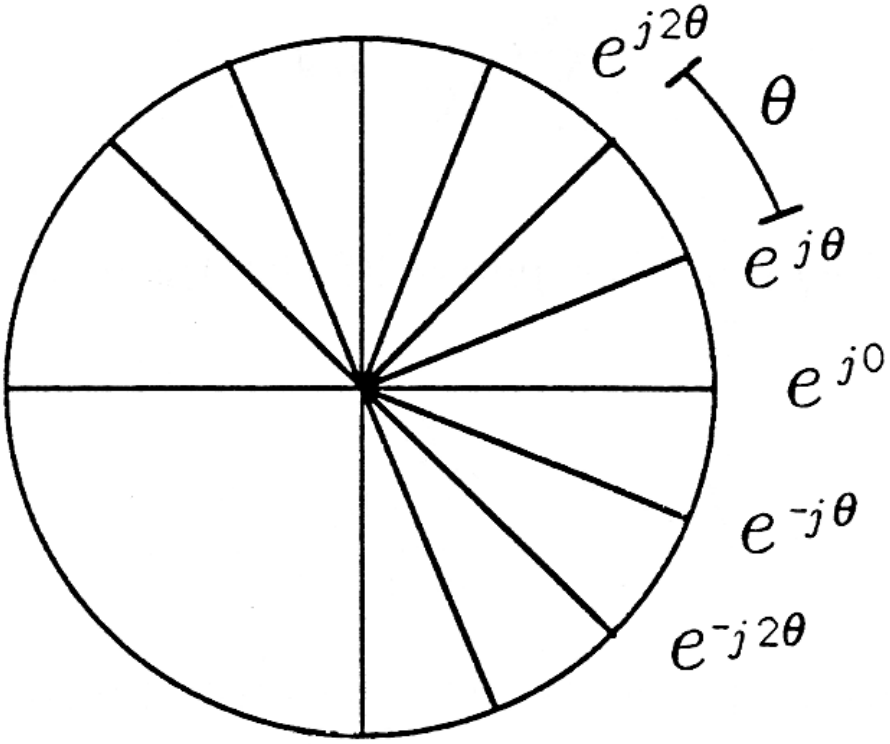
Find the unit step response for the recursive filter $x_n = a x_{n-1} + w_0 u_n$.

Complex Exponential Sequence. The complex exponential sequence is the sequence

Equation:

$$u_k = e^{jk\theta}, \;\; k = 0, \pm 1, \pm 2, ...$$

This sequence, illustrated in Figure 3, is a “discrete-time phasor” that “ratchets” counterclockwise (CCW) as k moves to $k+1$ and clockwise (CW) as k moves to $k-1$. Each time the phasor ratchets, it turns out an angle of θ. Why should such a sequence be a useful test sequence? There are two reasons.



Discrete-Time Phasor

(i) $e^{jk\theta}$ **represents (or codes)** cos $k\theta$. The real part of the sequence $e^{jk\theta}$ is the cosinusoidal sequence cos $k\theta$:
Equation:

$$\mathrm{Re}\left[e^{jk\theta}\right] = \cos\, k\theta.$$

Therefore the discrete-time phasor $e^{jk\theta}$ represents (or codes) cos $k\theta$ in the same way that the continuous-time phasor $e^{j\omega t}$ codes cos ωt. If the moving average filter
Equation:

$$x_n = \sum_{k=0}^{\infty} h_k u_{n-k}$$

has real coefficients, we can get the response to a cosinusoidal sequence by taking the real part of the following sum:
Equation:

$$\begin{aligned} x_n &= \sum_{k=0}^{\infty} h_k \cos{(n-k)\theta} \\ &= \mathrm{Re}\left[\sum_{k=0}^{\infty} h_k e^{j(n-k)\theta}\right] \\ &= \mathrm{Re}\left[e^{jn\theta} \sum_{k=0}^{\infty} h_k e^{-jk\theta}\right]. \end{aligned}$$

In this formula, the sum
Equation:

$$\sum_{k=0}^{\infty} h_k e^{-jk\theta}$$

is called the *complex frequency response* of the filter and is given the symbol
Equation:

$$H\left(e^{j\theta}\right) = \sum_{k=0}^{\infty} h_k e^{-jk\theta}.$$

This complex frequency response is just a complex number, with a magnitude $\left|H\left(e^{j\theta}\right)\right|$ and a phase arg $H\left(e^{j\theta}\right)$. Therefore the output of the moving average filter is
Equation:

$$\begin{aligned} x_n &= \mathrm{Re}\left[e^{jn\theta} H\left(e^{j\theta}\right)\right] \\ &= \mathrm{Re}\left[e^{jn\theta} \left|H\left(e^{j\theta}\right)\right| e^{j\arg H\left(e^{j\theta}\right)}\right] \\ &= \left|H\left(e^{j\theta}\right)\right| \ cos \ \left[n\theta + \arg H\left(e^{j\theta}\right)\right]. \end{aligned}$$

This remarkable result says that the output is also cosinusoidal, but its amplitude is $\left|H\left(e^{j\theta}\right)\right|$ rather than 1, and its phase is $argH\left(e^{j\theta}\right)$ rather than 0. In the examples to follow, we will show that the complex "gain" $H\left(e^{j\theta}\right)$ can be highly selective in θ, meaning that cosines of some angular frequencies are passed with little attenuation while cosines of other frequencies are dramatically attenuated. By choosing the filter coefficients, we can design the frequency selectivity we would like to have.

(ii) $e^{jk\theta}$ is a sampled data version of $e^{j\omega t}$. The discrete-time phasor $e^{jk\theta}$ can be produced physically by sampling the continuous-time phasor $e^{j\omega t}$ at the periodic sampling instants $t_k = kT$:
Equation:

$$e^{jk\theta} = e^{j\omega t}\Big|_{t=kT} = e^{j\omega kT}$$
$$\theta = \omega T.$$

The dimensions of θ are radians, the dimensions of ω are radians/second, and the dimensions of T are seconds. We call T the sampling interval and $\frac{1}{T}$ the sampling rate or sampling frequency. If the original angular frequency of the phasor $e^{j\omega t}$ is increased to $\omega + m\left(\frac{2\pi}{T}\right)$, then the discrete-time phasor remains $e^{jk\theta}$:
Equation:

$$e^{j[\omega+m(2\pi/T)]t}\Big|_{t=kT} = e^{j(\omega kT+km2\pi)} = e^{jk\theta}.$$

This means that all continuous-time phasors of the form $e^{j[\omega+m(2\pi/T)]t}$ "hide under the same alias" when viewed through the sampling operation. That is, the sampled-data phasor cannot distinguish the frequency ω from the frequency $\omega + m\frac{2\pi}{T}$. In your subsequent courses you will study aliasing in more detail and study the Nyquist rule for sampling:
Equation:

$$T \leq \frac{2\pi}{\Omega} \;\; ; \;\; \frac{1}{T} \geq \frac{\Omega}{2\pi}$$

This rule says that you must sample signals at a rate $\left(\frac{1}{T}\right)$ that exceeds the *bandwidth* $\frac{\Omega}{2\pi}$ of the signal.

Example:
Let's pass the cosinusoidal sequence $u_k = \cos k\theta$ through the finite moving average filter
Equation:

$$x_n = \sum_{k=0}^{N-1} h_k u_{n-k}$$
$$h_k = \tfrac{1}{N} \;\; k = 0, 1, \;\; ..., \;\; N-1.$$

We know from our previous result that the output is
Equation:

$$x_n = \left|H\left(e^{j\theta}\right)\right| \cos\left[n\theta + \arg H\left(e^{j\theta}\right)\right].$$

The complex frequency response for this example is
Equation:

$$\begin{aligned} H\left(e^{j\theta}\right) &= \sum_{k=0}^{N-1} \tfrac{1}{N} e^{-jk\theta} \\ &= \tfrac{1}{N} \tfrac{1-e^{-jN\theta}}{1-e^{-j\theta}} \end{aligned}$$

(Do you see your old friend, the finite sum formula, at work?) Let's try to manipulate the result into a more elegant form:
Equation:

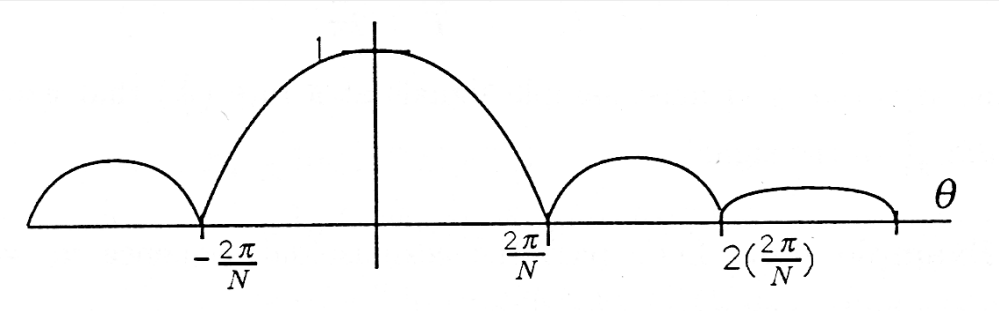
$$
\begin{aligned}
H\left(e^{j\theta}\right) &= \frac{1}{N}\frac{e^{-j(N/2)\theta}\left[e^{j(N/2)\theta}-e^{-j(N/2)\theta}\right]}{e^{-j(\theta/2)}\left[e^{j(\theta/2)}-e^{-j(\theta/2)}\right]} \\
&= \frac{1}{N}e^{-j[(N-1)/2]\theta}\frac{\sin\left(\frac{N}{2}\theta\right)}{\sin\left(\frac{1}{2}\theta\right)}.
\end{aligned}
$$

The magnitude of the function $H\left(e^{j\theta}\right)$ is

Equation:

$$
|H\left(e^{j\theta}\right)| = \frac{1}{N}, \left|\frac{\sin\left(\frac{N}{2}\theta\right)}{\sin\left(\frac{1}{2}\theta\right)}\right|.
$$

At $\theta = 0$, corresponding to a "DC phasor," $H\left(e^{j\theta}\right)$ equals 1; at $\theta = \frac{2\pi}{N}$ $\left|H\left(e^{j\theta}\right) = 0\right|$. The magnitude of the complex frequency response is plotted in Figure 4.



Frequency Selectivity of a Moving Average Filter

This result shows that the moving average filter is frequency selective, passing low frequencies with gain near 1 and high frequencies with gain near 0.

Exercise:

Problem: Compute the phase of the complex frequency response
Equation:

$$H\left(e^{j\theta}\right) = \frac{1}{N} e^{-j[(N-1)/2]\theta} \frac{\sin\left(\frac{N}{2}\theta\right)}{\sin\left(\frac{1}{2}\theta\right)}.$$

Exercise:

Problem:

Choose the filter length N for the filter $h_k = \frac{1}{N}, k = 0, 1, ..., N-1$, so that a 60 Hz cosine, sampled at the rate $\frac{1}{T} = 180$, is perfectly zeroed out as it comes through the filter.

Exercise:

Problem:

(MATLAB) Write a MATLAB program to compute and plot the magnitude $\left|H\left(e^{j\theta}\right)\right|$ and the phase $\arg H\left(e^{j\theta}\right)$ versus $-\pi < \theta < \pi$ when

Equation:

$$H\left(e^{j\theta}\right) = \frac{1}{N} e^{-j[(N-1)/2]\theta} \frac{\sin\left(\frac{N}{2}\theta\right)}{\sin\left(\frac{1}{2}\theta\right)}.$$

Choose suitable increments for θ.

Exercise:

Problem:

Compute the complex frequency response $H\left(e^{j\theta}\right)$ for the recursive filter $x_n = ax_{n-1} + w_0 u_n$.

Filtering: Numerical Experiment (Frequency Response of First-Order Filter)

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Consider the exponential moving average filter
Equation:

$$x_n = \sum_{k=0}^{\infty} a^k u_{n-k;} \;\; a = 0.98.$$

1. Write out a few terms of the sum to show how the filter works.
2. Write x_n as a recursion and discuss the computer memory required to implement the filter.
3. Compute the complex frequency response $H\;\; e^{j\theta}$ for the filter.
4. Write a MATLAB program to plot the magnitude and phase of the complex frequency response $H\;\; e^{j\theta}$ versus θ for $\theta = -\pi \text{to} \; + \pi$ in steps of $\frac{2\pi}{64}$ Do this for two values of a, namely, $a = 0.98$ and $a = -0.98$. Explain your findings.
5. Write a MATLAB program to pass the following signals through the filter when $a = 0.98$:

   a. $u_n = \delta_n$
   b. $u_n = \xi_n$
   c. $u_n = \xi_n \cos \frac{2\pi}{64} n$

d. $u_n = \xi_n \cos \dfrac{2\pi}{32} n$

e. $u_n = \xi_n \cos \dfrac{2\pi}{16} n$

f. $u_n = \xi_n \cos \dfrac{2\pi}{8} n$

g. $u_n = \xi_n \cos \dfrac{2\pi}{4} n$

h. $u_n = \xi_n \cos \dfrac{2\pi}{2} n.$

Plot the outputs for each case and interpret your findings in terms of the complex frequency response $H\ \ e^{j\theta}$. Repeat step 5 for $a = -0.98$. Interpret your findings.

Binary Codes: Introduction

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Acknowledgment: Richard Hamming's book, *Information Theory andCoding*, Prentice-Hall, New York (1985) and C. T. Mullis's unpublished notes have influenced our treatment of binary codes. The numerical experiment was developed by Mullis.

We use this chapter to introduce students to the communication paradigm and to show how arbitrary symbols may be represented by binary codes. These symbols and their corresponding binary codes may be computer instructions, integer data, approximations to real data, and so on.

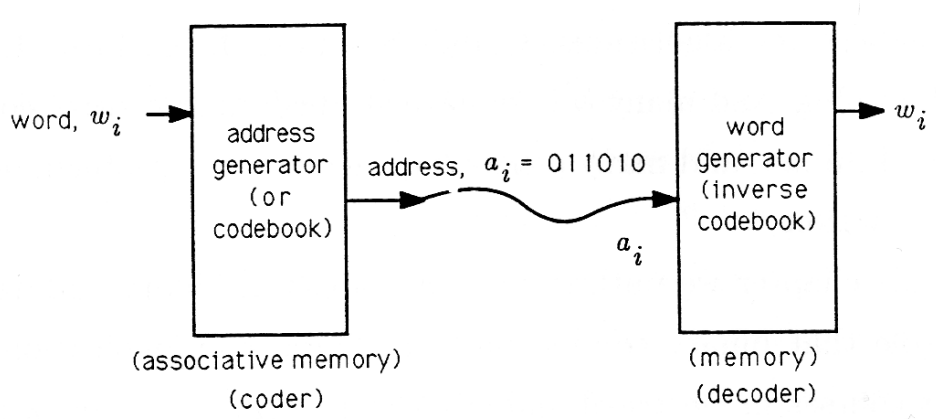
We develop some ad hoc tree codes for representing information and then develop Huffman codes for optimizing the use of bits. Hamming codes add check bits to a binary word so that errors may be detected and corrected. The numerical experiment has the students design a Huffman code for coding Lincoln's Gettysburg Address.

Introduction

It would be stretching our imagination to suggest that Sir Francis had digital audio on his minde (sic) when he wrote the prophetic words

"Sir Francis Bacon, 1623 ...a man may express and signifie the intentions of his minde, at any distance... by... objects... capable of a twofold difference onely. "

Nonetheless, this basic idea forms the basis of everything we do in digital computing, digital communications, and digital audio/video. In 1832, Samuel F. B. Morse used the very same idea to propose that telegram *words* be coded into *binary addresses* or *binary codes* that could be transmitted over telegraph lines and decoded at the receiving end to unravel the telegram. Morse abandoned his scheme, illustrated in Figure 1, as too complicated and, in 1838, proposed his fabled Morse code for coding letters (instead of words) into objects (dots, dashes, spaces) capable of a threefold difference onely (sic).



Generalized Coder-Decoder

The basic idea of Figure 1 is used today in cryptographic systems, where the “address a_i" is an encyphered version of a message w_i ; in vector quantizers, where the “address a_i" is the address of a close approximation to data w_i ; in coded satellite transmissions, where the “address a_i" is a data word w_i plus parity check bits for detecting and correcting errors; in digital audio systems, where the “address a_i" is a stretch of digitized and coded music; and in computer memories, where a_i is an address (a coded version of a word of memory) and w_i is a word in memory.

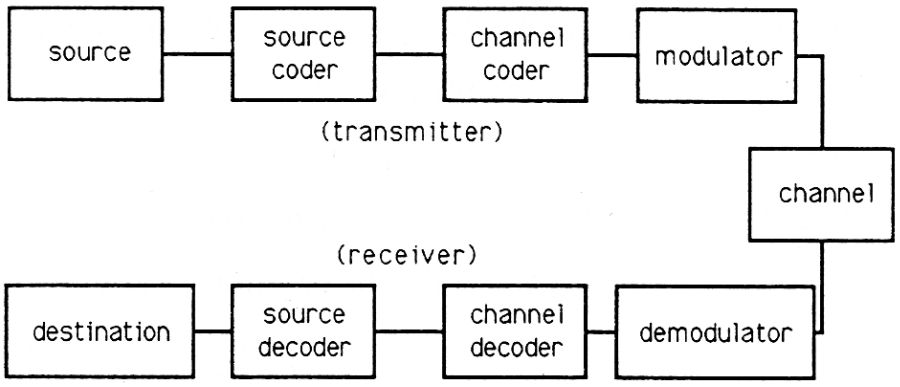
In this chapter we study three fundamental questions in the construction of binary addresses or binary codes. First, what are plausible schemes for mapping symbols (such as words, letters, computer instructions, voltages, pressures, etc.) into binary codes? Second, what are plausible schemes for coding likely symbols with short binary words and unlikely symbols with long words in order to minimize the number of binary digits (bits) required to represent a message? Third, what are plausible schemes for "coding" binary words into longer binary words that contain "redundant bits" that may be used to detect and correct errors? These are not new questions. They have occupied the minds of many great thinkers. Sir Francis recognized that arbitrary messages had binary representations. Alan Turing, Alonzo Church, and Kurt Goedel studied binary codes for computations in their study of computable numbers and algorithms. Claude Shannon, R. C. Bose, Irving Reed, Richard Hamming, and many others have studied error control codes. Shan- non, David Huffman, and many others have studied the problem of efficiently coding information.

In this chapter we outline the main ideas in binary coding and illustrate the role that binary coding plays in digital communications. In your subsequent courses in electrical and computer engineering you will study integrated circuits for building coders and decoders and mathematical models for designing good codes.

Binary Codes: The Communication Paradigm

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

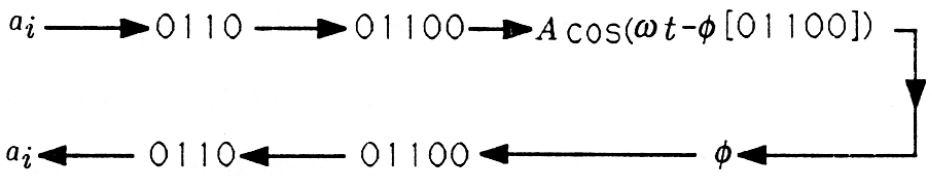
A *paradigm* is a pattern of ideas that form the foundation for a body of knowledge. A paradigm for (tele-) communication theory is a pattern of basic building blocks that may be applied to the dual problems of (i) reliably transmitting information from source to receiver at high speed or (ii) reliably storing information from source to memory at high density. High-speed communication permits us to accommodate many low-rate sources (such as audio) or one high-rate source (such as video). High-density storage permits us to store a large amount of information in a small space. For example, a typical 1.2 Mbyte floppy disc stores 9.6×10^6 bits of information, whereas a typical CD stores about 2×10^9 bits, enough for one hour's worth of high-quality sound.



Basic Building Blocks in a (Tele-) Communication System

Figure 1 illustrates the basic building blocks that apply to any problem in the theory of (tele-) communication. The *source* is an arbitrary source of information. It can be the time-varying voltage at the output of a vibration sensor (such as an integrating accelerometer for measuring motion or a microphone for measuring sound pressure); it can be the charges stored in the CCD array of a solid-state camera; it can be the addresses generated from a sequence of keystrokes at a computer terminal; it can be a sequence of instructions in a computer program. The *source coder* is a device for turning primitive source outputs into more efficient representations. For example, in a recording studio, the source coder would convert analog voltages into digital approximations using an A/D converter; a fancy source coder would use a fancy A/D converter that finely quantized likely analog values and crudely quantized unlikely values. If the source is a source of discrete symbols like letters and numbers, then a fancy source code would assign short binary sequences to likely symbols (such as e) and long binary sequences to unlikely symbols (such as z). The *channel coder* adds “redundant bits” to the binary output of the source coder so that errors of transmission or storage may be detected and corrected. In the simplest example, a binary string of the form 01001001 would have an extra bit of 1 added to give *even parity* (an even number of l's) to the string; the string 10110111 would have an extra bit of 0 added to preserve the *even parity*. If one bit error is introduced in the channel, then the parity is odd and the receiver knows that an error has occurred. The *modulator* takes outputs of the channel coder, a stream of binary digits, and constructs an analog waveform that represents a block of bits. For example, in a 9600 baud Modem, five bits are used to determine one of $2^5 = 32$ phases that are used to modulate the signal $A\ \cos\ (\omega t + \varphi)$. Each possible string of five bits has its own personalized phase, φ, and this phase can be determined at the receiver. The signal $A\ \cos\ (\omega t + \varphi)$ is an analog signal that may be transmitted over a *channel* (such as a telephone line, a microwave link, or a fiber-optic cable). The channel has a finite bandwidth, meaning that it distorts signals, and it is subject to noise or interference from other electromagnetic radiation. Therefore transmitted information arrives at the *demodulator* in imperfect form. The demodulator uses *filters* matched to the modulated signals to demodulate the phase and look up the corresponding bit stream. The *channel decoder* converts the coded bit stream into the

information bit stream, and the *source decoder* looks up the corresponding symbol. This sequence of steps is illustrated symbolically in Figure 7.3.



Symbolic Representation of Communication

In your subsequent courses on communication theory you will study each block of Figure 1 in detail. You will find that every source of information has a characteristic complexity, called *entropy*, that determines the minimum rate at which bits must be generated in order to represent the source. You will also find that every communication channel has a characteristic tolerance for bits, called *channel capacity*. This capacity depends on signal-to-noise ratio and bandwidth. When the channel capacity exceeds the source entropy, then you can transmit information reliably; if it does not, then you cannot.

Binary Codes: From Symbols to Binary Codes

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Perhaps the most fundamental idea in communication theory is that arbitrary symbols may be represented by strings of binary digits. These strings are called binary words, binary addresses, or binary codes. In the simplest of cases, a finite alphabet consisting of the letters or symbols $s_0, s_1, ..., s_{M-1}$ is represented by binary codes. The obvious way to implement the representation is to let the ith binary code be the binary representation for the subscript i:
Equation:

$$\begin{aligned} s_0 \sim 000 &= a_0 \\ s_1 \sim 001 &= a_1 \\ &\vdots \\ s_6 \sim 110 &= a_6 \\ s_7 \sim 111 &= a_7. \end{aligned}$$

The number of bits required for the binary code is N where
Equation:

$$2^{N-1} < M \leq 2^N.$$

We say, roughly, that $N = \log_2 M$.

Octal Codes. When the number of symbols is large and the corresponding binary codes contain many bits, then we typically group the bits into groups

of three and replace the binary code by its corresponding octal code. For example, a seven-bit binary code maps into a three-digit octal code as follows:

Equation:

$$\begin{aligned} 0000000 &\sim 000 \\ 0000001 &\sim 001 \\ &\vdots \\ 0100110 &\sim 046 \\ &\vdots \\ 101111 &\sim 137 \\ &\vdots \\ 1111111 &\sim 177. \end{aligned}$$

The octal ASCII codes for representing letters, numbers, and special characters are tabulated in Table 1.

Exercise:

Problem: Write out the seven-bit ASCII codes for $A, q, 7, and \{$.

	'0	*'1*	*'2*	*'3*	*'4*	*'5*	*'6*	*'7*
'00x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL
'01x	BS	HT	LF	VT	FF	CR	SO	SI
'02x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB
'03x	CAN	EM	SUB	ESC	FS	GS	RS	US

'04x	SP	!	"	#	$	%	&	'
'05x	(	)	*	+	,	-	.	/
'06x	0	1	2	3	4	5	6	7
'07x	8	9	:	;	<	=	>	?
'10x	@	A	B	C	D	E	F	G
'11x	H	I	J	K	L	M	N	O
'12x	P	Q	R	S	T	U	V	W
'13x	X	Y	Z	[	\	]	^	_
'14x	`	a	b	c	d	e	f	g
'15x	h	i	j	k	l	m	n	o
'16x	p	q	r	s	t	u	v	w
'17x	x	y	z	{	|	}	~	DEL

Octal ASCII Codes (from Donald E. Knuth, *The TEXbook*, ©1986 by the American Mathematical Society, Providence, Rhode Island p. 367, published by Addison-Wesley Publishing Co.)

Exercise:

Problem:

Add a 1 or a 0 to the most significant (left-most) position of the seven-bit ASCII code to produce an eight-bit code that has even parity (even number of 1's). Give the resulting eight-bit ASCII codes and the corresponding three-digit octal codes for %, u, f, 8, and +.

Quantizers and A/D Converters. What if the source alphabet is infinite? Our only hope is to approximate it with a finite collection of finite binary words. For example, suppose the output of the source is an analog voltage that lies between $-V_0$ and $+V_0$. We might break this peak-to-peak range up into little voltage cells of size $\frac{2V}{M}\,\mathrm{A}$ and approximate the voltage in each cell by its midpoint. This scheme is illustrated in Figure 1. In the figure, the cell C_i is defined to be the set of voltages that fall between $i\frac{2VpV_{\Delta}}{M^{-}M}$ and $i^{\underline{2V}}M\Delta + \frac{V}{-}M^{\mathrm{A}}$:

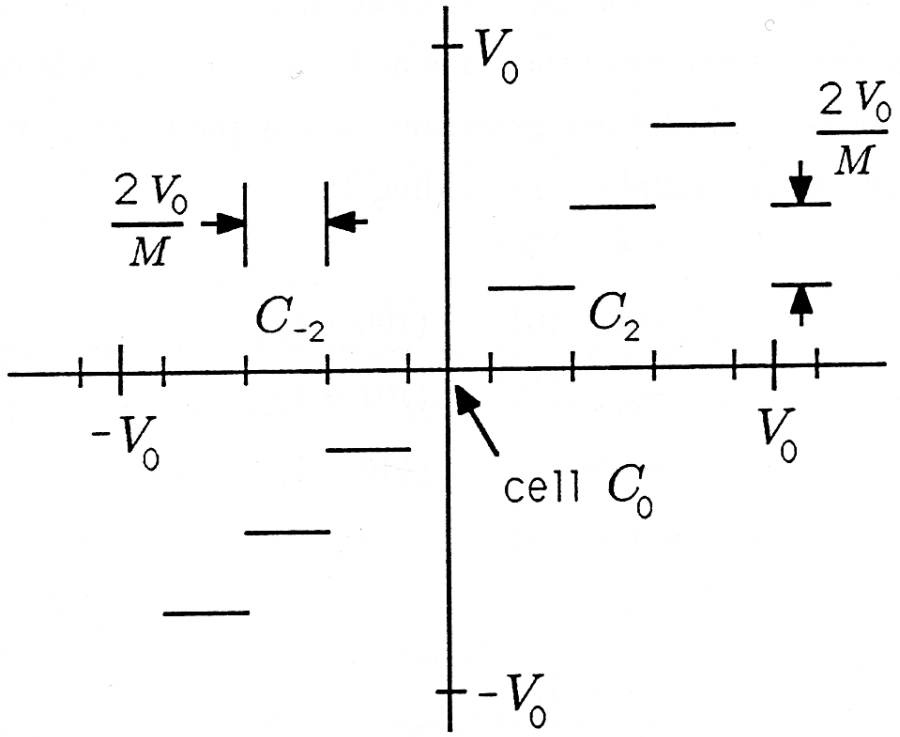
Equation:

$$C_i = \left\{ V : i.\,\frac{2V_0}{M} - \frac{V_0}{M} < V \leq i\frac{2V_0}{M} + \frac{V_0}{M} \right\}.$$

The mapping from continuous values of V to a finite set of approximations is
Equation:

$$Q\left(V\right) = i\frac{2V_0}{M}, \;\; \text{if} \;\; V \in C_i.$$

That is, V is replaced by the quantized approximation $i\frac{2V_0}{M}$ whenever V lies in cell C_i. We may represent the quantized values $i\frac{2V}{M}\,0$ with binary codes by simply representing the subscript of the cell by a binary word. In a subsequent course on digital electronics and microprocessors you will study $\mathrm{A/D}$ (analog-to-digital) converters for quantizing variables.



A Quantizer

Example:
If $M = 8$, corresponding to a three-bit quantizer, we may associate quantizer cells and quantized levels with binary codes as follows:
Equation:

$$
\begin{aligned}
V \in C_{-3} &\Rightarrow V_{-3} = (-3)\tfrac{2V_0}{8} \sim 111 \\
V \in C_{-2} &\Rightarrow V_{-2} = (-2)\tfrac{2V_0}{8} \sim 110 \\
V \in C_{-1} &\Rightarrow V_{-1} = (-1)\tfrac{2V_0}{8} \sim 101 \\
V \in C_{0} &\Rightarrow V_{0} = 0 \sim 000 \\
V \in C_{1} &\Rightarrow V_{1} = (1)\tfrac{2V_0}{8} \sim 001 \\
V \in C_{2} &\Rightarrow V_{2} = (2)\tfrac{2V_0}{8} \sim 010 \\
V \in C_{3} &\Rightarrow V_{3} = (3)\tfrac{2V_0}{8} \sim 011.
\end{aligned}
$$

This particular code is called a *sign-magnitude code*, wherein the most significant bit is a sign bit and the remaining bits are magnitude bits (e.g., $110 \sim -2$ and $010 \sim 2$). One of the defects of the sign-magnitude code is that it wastes one code by using 000 for 0 and 100 for-O. An alternative code that has many other advantages is the 2's *complement code*. The $2'$s complement codes for positive numbers are the same as the sign-magnitude codes, but the codes for negative numbers are generated by complementing all bits for the corresponding positive number and adding 1:

Equation:

$$
\begin{aligned}
-4 &\sim 100 \\
-3 &\sim 101 \quad (100+1) \\
-2 &\sim 110 \quad (101+1) \\
-1 &\sim 111 \quad (110+1) \\
0 &\sim 000 \\
1 &\sim 001 \\
2 &\sim 010 \\
3 &\sim 011.
\end{aligned}
$$

Exercise:

Problem:

Generate the four-bit sign-magnitude and four-bit 2's complement binary codes for the numbers $-8, -7, ..., -1, 0, 1, 2, ..., 7$.

Exercise:

Problem:

Prove that, in the $2'$s complement representation, the binary codes for $-n \text{and} + n$ sum to zero. For example,

Equation:

$$\begin{gathered} 101 + 011 = 000 \\ (-3)\ (3)\ (0). \end{gathered}$$

In your courses on computer arithmetic you will learn how to do arithmetic in various binary-coded systems. The following problem illustrates how easy arithmetic is in 2's complement.

Exercise:

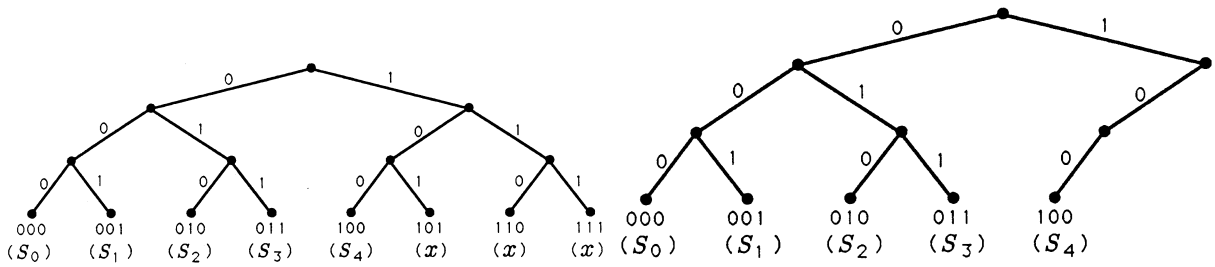
Problem:

Generate a table of sums for all $2'$s complement numbers between -4 and +3. Show that the sums are correct. Use $0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1$, and $1 + 1 = 0$ with a carry into the next bit. For example, $001 + 001 = 010$.

Binary Trees and Variable-Length Codes. The codes we have constructed so far are constant-length codes for finite alphabets that contain exactly $M = 2^N$ symbols. In the case where $M = 8$ and $N = 3$, then the eight possible three-bit codes may be represented as leaves on the branching tree illustrated in Figure 2(a). The tree grows a left branch for a 0 and a right branch for a 1, until it terminates after three branchings. The three-bit codes we have studied so far reside at the terminating leaves of the binary tree. But what if our source alphabet contains just five symbols or letters? We can represent these five symbols as the three-bit symbols 000 through 100 on the binary tree. This generates a constant-length code with three unused, or

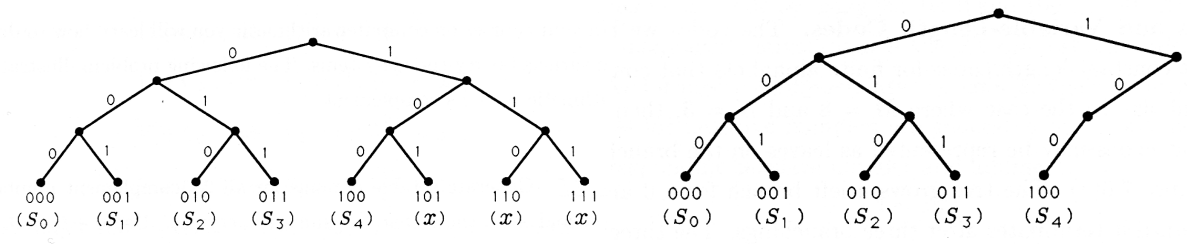
illegal, symbols 101 through 111. These are marked with an "x" in Figure 2(a). These unused leaves and the branches leading to them may be pruned to produce the binary tree of Figure 2(b).

If we admit variable-length codes, then we have several other options for using a binary tree to construct binary codes. Two of these codes and their corresponding binary trees are illustrated in Figure 3. If we disabuse ourselves of the notion that each code word must contain three or fewer bits, then we may construct binary trees like those of Figure 4 and generate their corresponding binary codes. In Figure 4(a), we grow a right branch after each left branch and label each leaf with a code word. In Figure 4(b), we prune off the last right branch and associatea code word with the leaf on the last left branch.



Binary Trees and Constant-Length Codes; (a) Binary Tree, and (b) Pruned Binary Tree

All of the codes we have generated so far are organized in Table 2. For each code, the average number of bits/symbol is tabulated. This average ranges from 2.4 to 3.0. If all symbols are equally likely to appear, then the best variable-length code would be code 2.



Binary Trees and Variable-Length Codes; (a) Binary Tree for Variable-length Code, and (b) Another Binary Tree for Variable-length Code

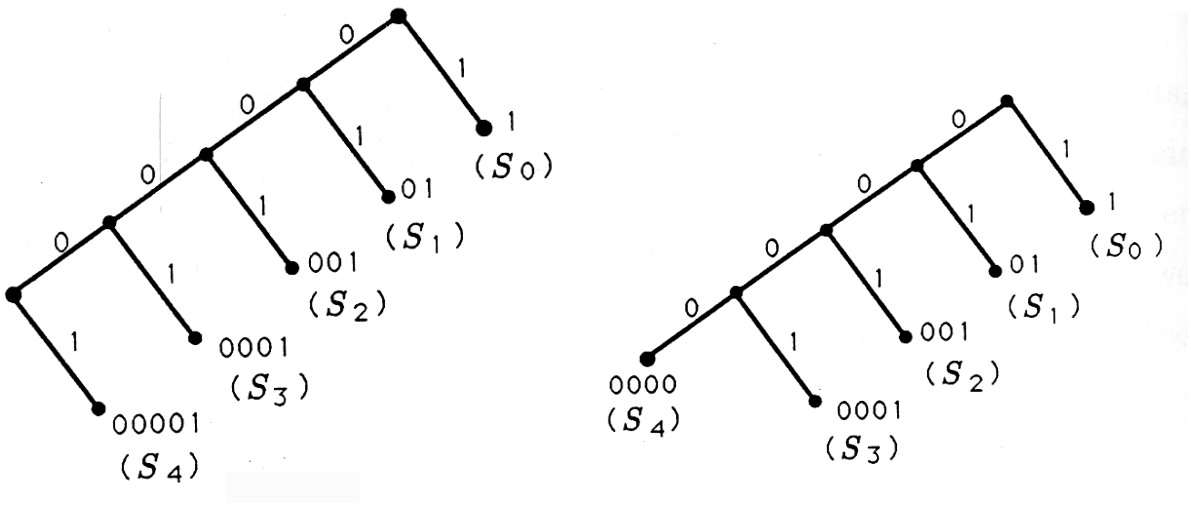
All of the codes we have constructed have a common characteristic: each code word is a terminating leaf on a binary tree, meaning that no code word lies along a limb of branches to another code word. We say that no code word is a *prefix* to another code word. This property makes each of the codes *instantaneously decodable*, meaning that each bit in a string of bits may be processed instantaneously (or independently) without dependence on subsequent bits.

Exercise:

Problem: Decode the following sequence of bits using code 2:

Equation:

$$01110011110000001011001111.$$



Left-Handed Binary Trees for Variable-Length Codes; (a) Left-handed Binary Tree, and (b) Pruned Binary Tree

Code #	S_0	S_1	S_2	S_3	S_4	**Average Bits/Symbol**
1	000	001	010	011	100	$15/5 = 3.0$
2	000	001	01	10	11	$12/5 = 2.4$
3	000	001	010	011	1	$13/5 = 2.6$
4	1	01	001	0001	00001	$15/5 = 3.0$
5	1	01	001	0001	0000	$14/5 = 2.8$

Variable Length Codes

Exercise:

Problem:

Illustrate the following codes on a binary tree. Which of them are instantaneously decodable? Which can be pruned and remain instantaneously decodable?

Equation:

$$\begin{array}{lllll} S_0 & S_1 & S_2 & S_3 & S_4 \\ 011 & 100 & 00 & 11 & 101 \\ 011 & 100 & 00 & 0 & 01 \\ 010 & 000 & 100 & 101 & 111. \end{array}$$

Code #2 generated in Table 2 seems like a better code than code #5 because its average number of bits/symbol (2.4) is smaller. But what if symbol S_0 is a very likely symbol and symbol S_4 is a very unlikely one? Then it may well turn out that the average number of bits used by code #5 is less than the average number used by code #2. So what is the best code? The answer depends on the relative frequency of use for each symbol. We explore this question in the next section.

Binary Codes: Huffman Codes for Source Coding

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

In 1838, Samuel Morse was struggling with the problem of designing an efficient code for transmitting information over telegraph lines. He reasoned that an efficient code would use short code words for common letters and long code words for uncommon letters. (Can you see the profit motive at work?) In order to turn this reasoned principle into workable practice, Morse rummaged around in the composition trays for typeface in a printshop. He discovered that typesetters use many more e's than s's. He then formed a table that showed the relative frequency with which each letter was used. His ingenious, variable-length Morse code assigned short codes to likely letters (like “dot” for e') and long codes to unlikely letters (like “dash dash dot dot” for z'). We now know that Morse came within about 15% of the theoretical minimum for the average code word length for English language text.

A Variation on Morse's Experiment. In order to set the stage for our study of efficient source codes, let's run a variation on Morse's experiment to see if we can independently arrive at a way of designing codes. Instead of giving ourselves a composition tray, let's start with a communication source that generates five symbols or letters S_0, S_1, S_2, S_3, S_4. We run the source for 100 transmissions and observe the following numbers of transmissions for each symbol:

Equation:

$$50\ S_0'\text{s}$$
$$20\ S_1'\text{s}$$
$$20 S_2'\text{s}$$
$$5\ S_3'\text{s}$$
$$5\ S_4'\text{s}.$$

We will assume that these “source statistics” are typical, meaning that 1000 transmissions would yield 500 S_0's and so on.

The most primitive binary code we could build for our source would use three bits for each symbol:
Equation:

$$S_0 \sim 000$$
$$S_1 \sim 001$$
$$S_2 \sim 010$$
$$S_3 \sim 011$$
$$S_4 \sim 100$$
$$x \sim 101$$
$$x \sim 110$$
$$x \sim 111.$$

This code is inefficient in two ways. First, it leaves three illegal code words that correspond to no source symbol. Second, it uses the same code word length for an unlikely symbol (like S_4) that it uses for a likely symbol (like S_0). The first defect we can correct by concatenating consecutive symbols into symbol blocks, or composite symbols. If we form a composite symbol consisting of M source symbols, then a typical composite symbol is $S_1S_0S_1S_4S_2S_3S_1S_2S_0$. The number of such composite symbols that can be generated is 5^M. The binary code for these 5^M composite symbols must contain N binary digits where
Equation:

$$2^{N-1} < 5^M < 2^N \, (N \cong M \log_2 5).$$

The number of bits per source symbol is
Equation:

$$\frac{N}{M} \cong \log_2 5 = 2.32.$$

This scheme improves on the best variable length code of Table 2 from "Binary Codes: From Symbols to Binary Codes" by 0.08 bits/symbol.
Exercise:

Problem:

Suppose your source of information generates the 26 lowercase roman letters used in English language text. These letters are to be concatenated into blocks of length *M*. Complete the following table of *N* (number of bits) versus *M* (number of letters in a block) and show that $\frac{N}{M}$ approaches $\log_2 26$.

	M					
	1	2	3	4	5	6
N	5	10				
N/M	5	5				

Now let's reason, as Morse did, that an efficient code would use short codes for likely symbols and long codes for unlikely symbols. Let's pick code #5 from Table 2 from "Binary Codes: From Symbols to Binary Codes" for this purpose:
Equation:

$$\begin{array}{ccccc} S_0 & S_1 & S_2 & S_3 & S_4 \\ 1 & 01 & 001 & 0001 & 0000. \end{array}$$

This is a variable-length code. If we use this code on the 100 symbols that generated our source statistic, the average number of bits/symbol is
Equation:

$$\frac{1}{100}\left[50\left(1\right)+20\left(2\right)+20\left(3\right)+5\left(4\right)+5\left(4\right)\right]=1.90\text{bits/symbol.}$$

Exercise:

> **Problem:**
>
> Use the source statistics of Equation 1 to determine the average number of bits/symbol for each code in Table 2 from "Binary Codes: From Symbols to Binary Codes".

Entropy. So far, each ad hoc scheme we have tried has produced an improvement in the average number of bits/symbol. How far can this go? The answer is given by Shannon's source coding theorem, which says that the minimum number of bits/symbol is
Equation:

$$\frac{N}{M}\geq-\sum_{i=1}^{M}p_i\log_2 p_i$$

where p_i is the probability that symbol S_i is generated and $-\sum_{i=1}^{M}p_i\log_2 p_i$ is a fundamental property of the source called *entropy*. For our five-symbol

example, the table of p_i and $-\log p_i$ is given in Table 2. The entropy is 1.861, and the bound on bits/symbol is

Equation:

$$\frac{N}{M} \geq 1.861.$$

Code $\#5$ comes within 0.039 of this lower bound. As we will see in the next paragraphs, this is as close as we can come without coding composite symbols.

Symbol	Probability	~ Log Probability
S_0	0.5	1
S_1	0.2	2.32
S_2	02	2.32
S_3	0.05	4.32
S_4	0.05	4.32

Source statistics for Five-Symbol Source

Exercise:

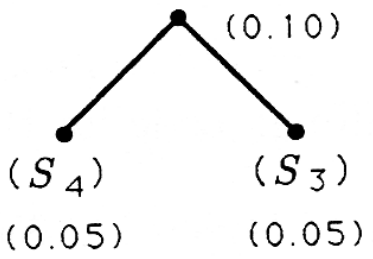
Problem:

Select an arbitrary page of English text. Build a table of source statistics containing p_i (relative frequencies) and-log p_i for a through z . (Ignore distinction between upper and lower case and ignore punctuation and other special symbols.) Compute the entropy

$$-\sum_{i=1}^{26} p_i \log_2 p_i.$$

Huffman Codes. In the late 1950s, David Huffman discovered an algorithm for designing variable-length codes that minimize the average number of bits/symbol. Huffman's algorithm uses a principle of optimality that says, “the optimal code for M letters has imbedded in it the optimal code for the $M-1$ letters that result from aggregating the two least likely symbols.” When this principle is iterated, then we have an algorithm for generating the binary tree for a Huffman code:

i. label all symbols as “children”;
ii. “twin” the two least probable children and give the twin the sum of the probabilities:



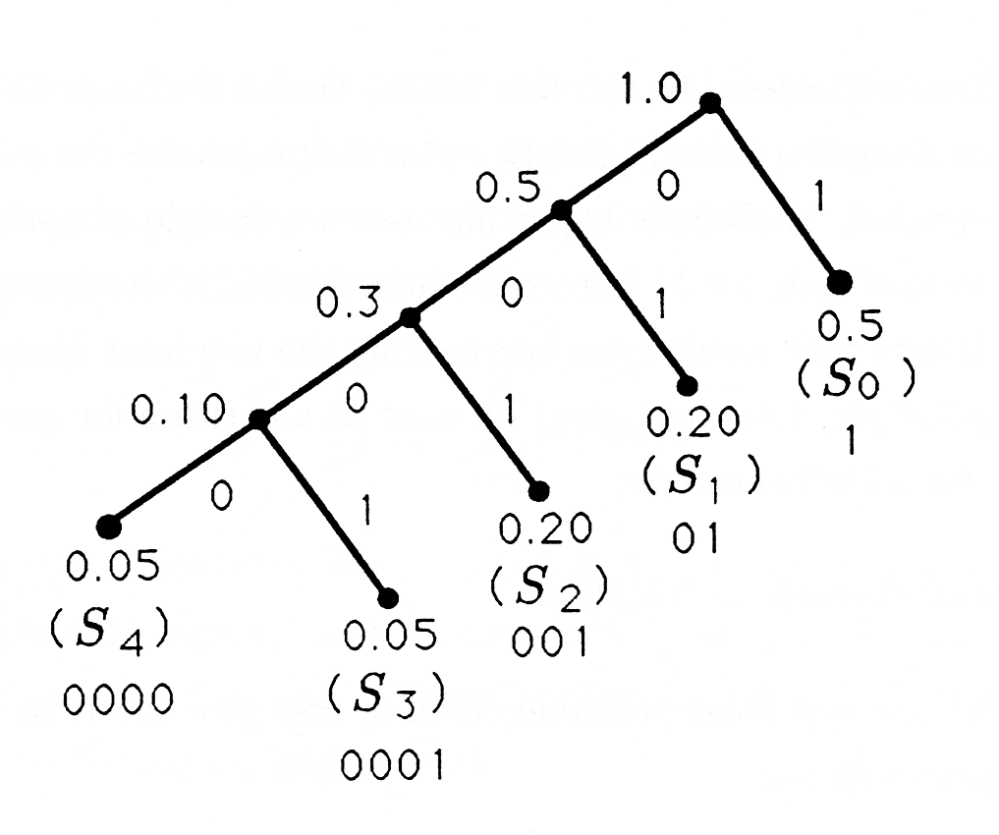
iii. (regard the twin as a child; and
iv. repeat steps (ii) and (iii) until all children are accounted for.

This tree is now labeled with 1's and 0's to obtain the Huffman code. The labeling procedure is to label each right branch with a 1 and each left branch with a 0. The procedure for laying out symbols and constructing Huffman trees and codes is illustrated in the following examples.

Example:
Consider the source statistics
Equation:

Symbol	S_0	S_1	S_2	S_3	S_4
Probability	0.5	0.2	0.2	0.05	0.05

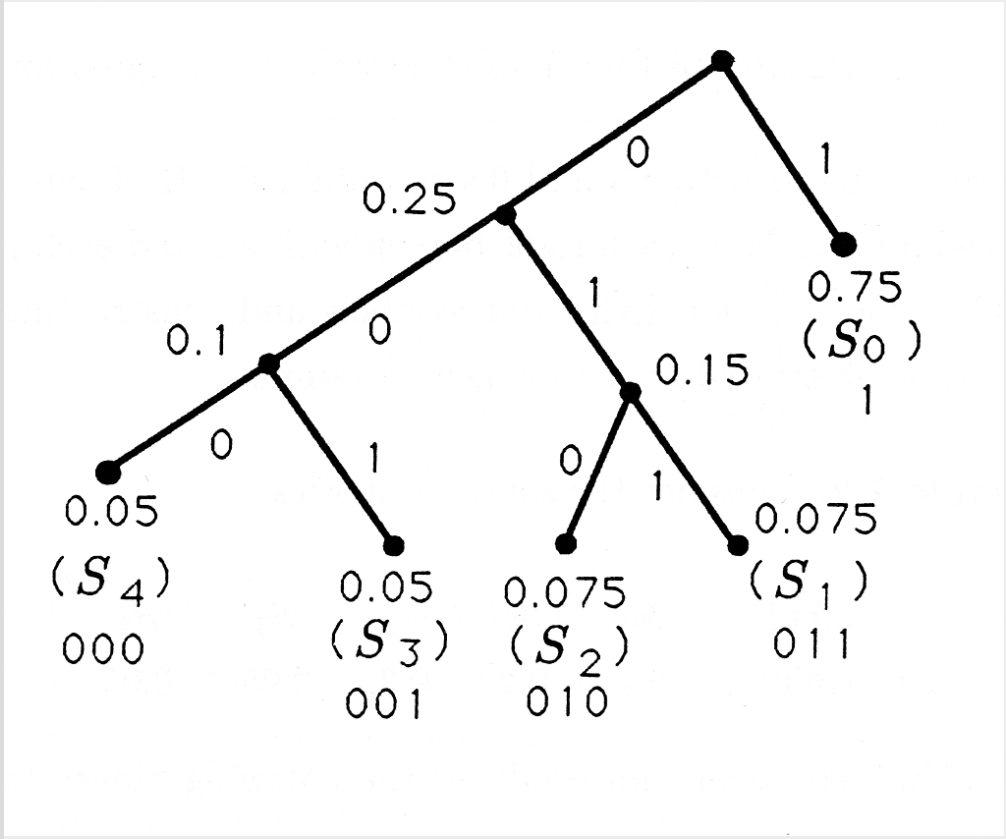
for which the Huffman algorithm produces the following binary tree and its corresponding code:



Example:
The Huffman code for the source statistics
Equation:

Symbol	S_0	S_1	S_2	S_3	S_4
Probability	0.75	0.075	0.075	0.05	0.05

is illustrated next:



Exercise:

Problem:

Generate binary trees and Huffman codes for the following source statistics:

Equation:

Symbol	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Probability1	0.20	0.20	0.15	0.15	0.1	0.1	0.05	0.05
Probability2	0.3	0.25	0.1	0.1	0.075	0.075	0.05	0.05.

Coding a FAX Machine. Symbols can arise in unusual ways and be defined quite arbitrarily. To illustrate this point, we consider the design of a

hypothetical FAX machine. For our design we will assume that a laser scanner reads a page of black and white text or pictures, producing a high voltage for a black spot and a low voltage for a white spot. We will also assume that the laser scanner resolves the page at 1024 lines, with 1024 spots/line. This means that each page is represented by a two-dimensional array, or matrix, of pixels (picture elements), each pixel being 1 or 0. If we simply transmitted these l's and O's, then we would need $1024 \times 1024 = 1,059,576$ bits. If these were transmitted over a 9600 baud phone line, then it would take almost 2 minutes to transmit the FAX. This is a long time.

Let's think about a typical scan line for a printed page. It will contain long runs of 0's, corresponding to long runs of white, interrupted by short bursts of l's, corresponding to short runs of black where the scanner encounters a line or a part of a letter. So why not try to define a symbol to be “a run of k 0's" and code these runs? The resulting code is called a “run length code.” Let's define eight symbols, corresponding to run lengths from 0 to 7 (a run length of 0 is a 1):
Equation:

$$\begin{aligned} S_0 &= \text{run length of 0 zeros } (a\ 1) \\ S_1 &= \text{run length of 1 zero} \\ &\vdots \\ S_7 &= \text{run legnth of 7 zeros.} \end{aligned}$$

If we simply used a simple three-bit binary code for these eight symbols, then for each scan line we would generate anywhere from 3×1024 bits (for a scan line consisting of all 1's) to $3 \times 1024/7 \cong 400$ bits (for a scan line consisting of all 0's). But what if we ran an experiment to determine the relative frequency of the run lengths S_0 through S_7 and used a Huffman code to “run length encode” the run lengths? The following problem explores this possibility and produces an efficient FAX code.
Exercise:

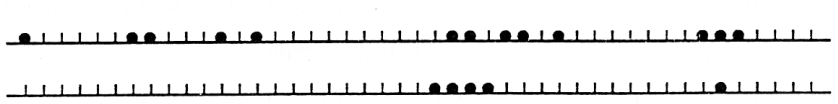
Problem:

An experiment conducted on FAXed documents produces the following statistics for run lengths of white ranging from 0 to 7:

Equation:

Symbol	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Probability	0.01	0.06	0.1	0.1	0.2	0.15	0.15	0.2

These statistics indicate that only 1% of a typical page is black. Construct the Huffman code for this source. Use your Huffman code to code and decode these scan lines:



Binary Codes: Hamming Codes for Channel Coding

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

The idea behind Hamming codes is to intersperse, or append, extra binary digits to a binary code so that errors in transmission of the code over a channel may be detected and corrected. For example, suppose we transmit the code 01101001, and it is received as 01001001. In this transmission, the third most significant bit is received erroneously. Let's define the following "modulo-2 addition" of binary numbers:

Equation:

$$\begin{aligned} 0 \oplus 0 &= 0 \\ 0 \oplus 1 &= 1 \\ 1 \oplus 0 &= 1 \\ 1 \oplus 1 &= 0. \end{aligned}$$

Multiplication in modulo-2 arithmetic is simply $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$ and $1 \cdot 1 = 1$. Then we can say that the *error sequence* 00100000 is "added" to the *transmission* 01101001 to produce the erroneous reception:

Equation:

$$\begin{array}{cll} & 01101001 & \text{transmitted} \\ \oplus & 00100000 & \text{error} \\ & 01001001 & \text{received.} \end{array}$$

Hamming error correcting codes will permit us to receive the erroneous transmission and to detect and correct the error. This is obviously of great value in transmitting and storing information. (Imagine how upset you would be to have the binary code for your checking account confused with that of Mrs. Joan Kroc.)

Choosing the Number of Check Bits. Let's suppose we have N bits of information that we wish to transmit and that we wish to intersperse "check bits" that will enable us to detect and correct any single bit error in the transmission. If we use N information bits and n check bits, then we will transmit a code word containing $N + n$ bits. The n check bits can code 2^n events, and we want these events to indicate whether or not any errors occurred and, if so, where they occurred. Therefore we require

where $(N + n)$ is the number of single error events that can occur and +1 is the number of no-error events. For example, when $N = 4$, we require $n = 3$ so that $2^3 \geq (4 + 3) + 1$.

How many check bits do you require to code seven bits of information for single error correction?

Code Construction. Let's suppose we have constructed an (N, n) Hamming code consisting of N information bits and n check bits (or parity bits). We denote the information bits by $x_1, x_2, ..., x_N$ and the check bits by $c_1, c_2, ..., c_n$. These bits may be interspersed. When $N = 4$ and $n = 3$, then a typical array of bits within a code word would be one of the following:
Equation:

$$\begin{array}{lcl} c_1 & & x_1 \\ c_2 & & x_2 \\ x_1 & & x_3 \\ c_3 & \text{or} & x_4 \\ x_2 & & c_1 \\ x_3 & & c_2 \\ x_4 & & c_3 \end{array} .$$

The first ordering is “natural” (as we will see), and the second is “systematic” (a term that is used to describe any code whose head is information and whose tail is check). If a single error occurs in an (N, n) code, then the received code word will be the modulo-2 sum of the code word and the error word that contains a 1 in its ith position:
Equation:

$$\begin{array}{lcl} c_1 & & 0 \\ c_2 & & 0 \\ x_1 & & 0 \\ c_3 & \oplus & 1 \\ x_2 & & 0 \\ x_3 & & 0 \\ x_4 & & 0 \end{array} .$$

We would like to operate on this received code word in such a way that the location of the error bit can be determined. If there were no code word, then an obvious solution would be to premultiply the error word by the *parity check* matrix
Equation:

$$A^T = \begin{array}{ccccccc} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ (1) & (2) & (3) & (4) & (5) & (6) & (7) \end{array} .$$

The ith column of A^T is just the binary code for i. When A^T premultiplies an error word, the error bit picks out the column that codes the error position:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

position 4 $\rightarrow$ $\uparrow$

binary code for position 4

If the error word contains no error bits, then the product is 0, indicating no errors.

This seems like a good idea, but what about the effect of the code word? In Exercise 1, you are asked to show that the effect of the parity check matrix A^T applied to the modulo-2 sum of a code word x and an error word e is
Equation:

$$\mathrm{A}^T(\mathrm{x} \oplus \mathrm{e}) = \mathrm{A}^T\mathrm{x} \oplus \mathrm{A}^T\mathrm{e}.$$

In this equation all sums and products obey the rules of modulo-2 arithmetic.
Exercise:

Problem:

Let $\mathrm{y} = \mathrm{x} \oplus \mathrm{e}$ denote the modulo-2 sum of a code word x and an error word e; A^T is a parity check matrix. Show that
Equation:

$$\mathrm{A}^T\mathrm{y} = \mathrm{A}^T\mathrm{x} \oplus \mathrm{A}^T\mathrm{e}.$$

We have designed the parity check matrix A^T so that the *syndrome* $\mathrm{A}^T\mathrm{e}$ produces a binary code for the error location. (The location of the error is

th′e syndrome for the error word.) The product $\mathrm{A}^T\mathrm{x}$ will interfere with this syndrome unless $A^T x = 0$. Therefore we will require that the code word x satisfy the constraint
Equation:

$$\mathrm{A}^T\mathrm{x} = 0.$$

This constraint actually *defines* the Hamming code. Let's illustrate this point by applying the constraint to a code word in its “natural format” $\mathrm{x}^T = (c_1 c_2 x_1 c_3 x_2 x_3 x_4)$.

Natural Codes. When the information bits and the check bits are coded in their natural order $(c_1 c_2 x_1 c_3 x_2 x_3 x_4)$, then we may determine the check bits by writing $\mathrm{A}^T\mathrm{x}$ as follows:
Equation:

$$\begin{array}{ccccccc} & & & & & & c_1 \\ & & & & & & c_2 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array} \begin{array}{c} x_1 \\ c_3 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \\ = \\ \\ \\ \\ \end{array} \begin{array}{c} 0 \\ 0 \\ 0 \\ \\ \\ \end{array}$$

We use the rules of modulo-2 arithmetic to write these constraints as
Equation:

$$\begin{aligned} c_1 \oplus x_1 \oplus x_2 \oplus x_4 &= 0 \\ c_2 \oplus x_1 \oplus x_3 \oplus x_4 &= 0 \\ c_3 \oplus x_2 \oplus x_3 \oplus x_4 &= 0. \end{aligned}$$

Therefore the check bits c_1, c_2, and c_3 are simply the following modulo-2 sums

Equation:

$$c_1 = x_1 \oplus x_2 \oplus x_4$$
$$c_2 = x_1 \oplus x_3 \oplus x_4$$
$$c_3 = x_2 \oplus x_3 \oplus x_4.$$

This finding may be organized into the matrix equation
Equation:

$$\begin{array}{cc} c_1 & \\ c_2 & \\ x_1 & \\ c_3 & = \\ x_2 & \\ x_3 & \\ x_4 & \end{array} \begin{array}{c} 1101 \\ 1011 \\ 1000 \\ 0111 \\ 0100 \\ 0010 \\ 0001 \end{array} \begin{array}{c} \\ \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} .$$

This equation shows how the code word x is built from the information bits (x_1, x_2, x_3, x_4). We call the matrix that defines the construction a *coder matrix* and write it as H:
Equation:

$$x = H\Theta$$
$$\mathrm{x}^T = (c_1 c_2 x_1 c_3 x_2 x_3 x_4) \ \Theta^T = (x_1 x_2 x_3 x_4)$$

Equation:

$$H = \begin{matrix} 1101 \\ 1011 \\ 1000 \\ 0111 \\ 0100 \\ 0010 \\ 0001 \end{matrix}$$

This summarizes the construction of a Hamming code x.

Exercise:

Problem:

Check to see that the product of the parity check matrix A^T and the coder matrix H is $\mathrm{A^T H} = 0$. Interpret this result.

Exercise:

Problem:

Fill in the following table to show what the Hamming $(4, 3)$ code is:

Equation:

x_1	x_2	x_3	x_4	c_1	c_2	x_1	c_3	x_2	x_3	x_4
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	1	0	0	1
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

Exercise:

Problem:

Design a Hamming $(11, n)$ code for coding eleven information bits against single errors. Show your equations for c_1, c_2, ..., c_n and write out the coder matrix H for $x = \mathrm{H}\Theta$.

Decoding. To decode a Hamming code, we form the syndrome $\mathrm{A}^T\mathrm{y}$ for the received (and possibly erroneous) code word $\mathrm{y} = \mathrm{x} \oplus \mathrm{e}$. Because $\mathrm{A}^T\mathrm{x} = 0$, the syndrome is
Equation:

$$\mathrm{s} = \mathrm{A}^T\mathrm{e}.$$

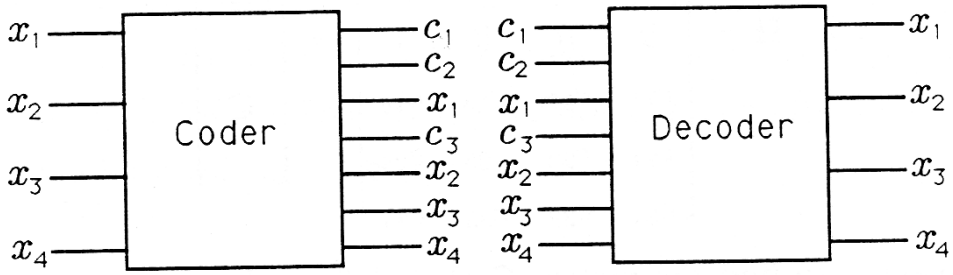
Convert this binary number into its corresponding integer location and change the bit of y in that location. If the location is zero, do nothing. Now strip off the information bits. This is the decoding algorithm.

Exercise:

Problem:

Use the table of Hamming $(4, 3)$ codes from Exercise 3 to construct a table of *received* codes that contain either no bit errors or exactly one bit error. Apply the decoding algorithm to construct (x_1, x_2, x_3, x_4) and show that all received code words with one or fewer errors are correctly decoded.

Digital Hardware. The tables you have constructed in Exercise 3 and 5 for coding and decoding Hamming $(4, 3)$ codes may be stored in digital logic chips. Their functionality is illustrated in Figure 1. The coder chip accepts $(x_1x_2x_3x_4)$ as its address and generates a coded word. The decoder chip accepts $(c_1c_2x_1c_3x_2x_3x_4)$ as its address and generates a decoded word. In your courses on digital logic you will study circuits for implementing coders and decoders.



Digital Logic for Hamming Code

Exercise:

Problem:

Discuss the possibility of detecting a received $(4, 3)$ code word that is neither a valid code word nor a code word with a single error. How would you use such a detector?

Exercise:

Problem:

What fraction of received seven-bit words can be correctly decoded as Hamming $(4, 3)$ codes?

Systematic Codes. Systematic Hamming codes are codes whose information bits lead and whose check bits trail. The format for a $(4, 3)$ code is then $(x_1 x_2 x_3 x_4 c_1 c_2 c_3)$. The construction of a $(4, 3)$ code word from the information bits may be written as

Equation:

$$\begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ c_1 \\ c_2 \\ c_3 \end{matrix} = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \end{matrix} \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{matrix}$$

The coder matrix takes the form

Equation:

$$H = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \end{matrix} = \begin{matrix} I \\ \hline\hline \\ C \end{matrix} .$$

The problem is to find the matrix C that defines the construction of check bits. The constraint $\mathrm{A}^T\mathrm{x} = 0$ produces the constraint $\mathrm{A}^T\mathrm{H} = 0$ so that $\mathrm{A}^T\mathrm{H}\Theta = 0$. The constraints $\mathrm{A}^T\mathrm{H} = 0$ may be written out as

Equation:

$$\begin{matrix} & & & & & & & \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{matrix} = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \end{matrix} = \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}.$$

These constraints produce all the equations we need (twelve equations in twelve unknowns) to determine the c_{ij}.

Exercise:

Problem:

Solve Equation 19 for the c_{ij}. Show that the coder matrix for a systematic Hamming $(4, 3)$ code is

Equation:

$$H = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{matrix}.$$

Exercise:

Problem:

Show that the coder matrix of Exercise 7 is a permutation of the coder matrix in Equation 14. (That is, the rows are reordered.)

Exercise:

Problem:

(MATLAB) Write a MATLAB program that builds Hamming $(4, 3)$ codes from information bits $(x_1x_2x_3x_4)$ and decodes Hamming $(4, 3)$ codes $(c_1c_2x_1c_3x_2x_3x_4)$ to obtain information bits $(x_1x_2x_3x_4)$. Synthesize all seven-bit binary codes and show that your decoder correctly decodes correct codes and one-bit error codes.

Binary Codes: Numerical Experiment (Huffman Codes)

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

The paragraph below contains Lincoln's Gettysburg Address. Ignore special symbols like periods and ignore the distinction between lowercase and uppercase letters. Compute the relative frequency of occurrence for each of the 26 letters of the alphabet. Write a MATLAB program to generate a binary tree for the Huffman code of the Gettysburg Address. When you have generated the code, compute the average code word length
Equation:

$$L = \sum_{i=1}^{26} \frac{n_i}{N} l_i$$

where $\frac{n_i}{N}$ is the relative frequency for symbol i and l_i is the code word length for symbol i. Compare L with the entropy
Equation:

$$H = -\sum_{i=1}^{26} \frac{n_i}{N} \log_2 \frac{n_i}{N} = \sum_{i=1}^{26} \frac{n_i}{N} \log_2 \frac{N}{n_i}$$

and compare L to $\log_2 26$. Interpret your findings.

Gettysburg Address

Fourscore and seven years ago, our fathers brought forth upon this
continent a new nation, conceived in liberty and dedicated to the proposition
that all men are created equal. Now we are engaged in a great civil war,
testing whether that nation or any nation so conceived and so dedicated can
long endure. We are met on a great battlefield of that war. We have come
to dedicate a portion of that field as a final resting-place for those who have
given their lives that that nation might live. It is altogether fitting and proper
that we should do this. But in a larger sense, we cannot dedicate, we cannot
consecrate, we cannot hallow this ground. The brave men, living and dead,
who struggled here have consecrated it far above our power to add or detract.
The world will little note nor long remember what we say here, but it can
never forget what they did here. It is for us the living rather to be dedicated
here to the unfinished work which they who fought here have thus far so nobly
advanced. It is rather for us to be here dedicated to the great task remaining
before us–that from these honored dead we take increased devotion to that
cause for which they gave the last full measure of devotion–that we here
highly resolve that these dead should not have died in vain, that this nation
under God shall have a new birth of freedom, and that government of the

people, by the people, for the people shall not perish from the Earth.

-Abraham Lincoln, November 19, 1863

An Introduction to MATLAB: Introduction

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Acknowledgment: This appendix, written with assistance from Cédric J. Demeure and Peter Massey, was inspired by the MATLAB user's manual from The MATHWORKS, Inc. The *MA TLAB Primer*, available through the MATLAB User's Group, is a useful learning aid for teachers and students. To join the MATLAB User's Group, send your request via E-mail to matlab-users request@mcs.anl.gov.

Introduction

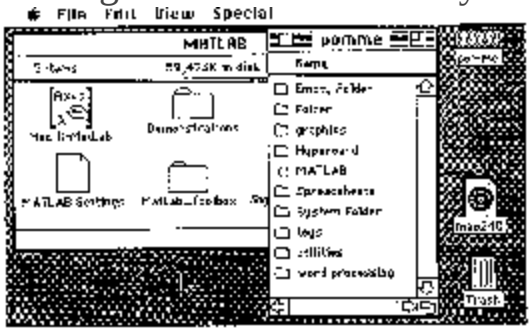
MATLAB stands for “Matrix Laboratory.” It is a computing environment specifically designed for matrix computations. The program is ideally suited to circuit analysis, signal processing, filter design, control system analysis, and much more. Beyond that, its versatility with complex numbers and graphics makes it an attractive choice for many other programming tasks. MATLAB can be thought of as a programming language like PASCAL, FORTRAN, C, or BASIC. Like most versions of BASIC, MATLAB can be used in an interactive mode wherein statements are executed immediately as they are typed. Alternatively, a program can be written in advance and saved to a disc file using an editor and then executed in MATLAB. You will find both modes of operation useful.

An Introduction to MATLAB: Running MATLAB (Macintosh)

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

In order to run MATLAB on a Macintosh SE or PLUS computer, you need the program called EDU-MATLAB. The program requires at least 1 Mbyte of memory, System 3.0 or above, Finder version 3.0 or above, and an 800K drive. A hard disc drive is highly recommended. In order to run MATLAB on a Macintosh II, IIx, IIcx, or SE/30, you need the program called MacII-MATLAB, and the same system requirements apply.

To start MATLAB, you may need to open the folder containing the MATLAB program. Then just "double-click" the program icon or the program name (for example, EDU-MATLAB). Figure A.1 shows a typical organization of the folder containing Mac II-MATLAB. It contains the main program, the settings file, the demonstrations folder, and any toolbox folders. The double-click on Mac II-MATLAB produces the *Command* window as shown in Figure A.2. You will also see a *Graph* window partially hidden behind it. (The fact that the window is not in ffont means that it is opened but not currently active.) If you do not know what "clicking," "dragging," "pop-up menu," and "trash" mean, you should stop reading now and familiarize yourself with the Macintosh.



The MATLAB Folder
(©Apple Computer, Inc.,
used with permission.)

In the command window, you should see the prompt ≫. The program interpreter is waiting for you to enter instructions. At this point it is a good idea to run the demonstration programs that are available in the “About MATLAB” menu under the Apple menu. Just click on the “demos” button and select a demo. During pauses, strike any key to continue. Whenever you have a MATLAB file in any folder, then you may double-click the file to launch the program. This allows you to have your own folder containing your own MATLAB files, separated from the MATLAB folder.



The Command Window
(©Apple Computer, Inc.,
used with permission.)

MATLAB has four types of windows:

i. *Command* for computing, programming, and designing input/output displays;
ii. *Graph* for displaying plots and graphs;
iii. *Edit* for creating and modifying your own files; and
iv. *Help* for getting on-line help and for running demos.

All windows follow the traditional behavior of Macintosh windows. You can resize them (actually the help window has a fixed size) or move them. For more details on menus and windows, see the Macintosh and MATLAB manuals.

An Introduction to MATLAB: Running MATLAB (PC)

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

In order to run MATLAB (Version 3.5) on an IBM or compatible personal computer, you must have a floating point math coprocessor installed and at least 512 kbytes of memory. The program is called PCMAT-LAB.EXE, but it is usually invoked via the batch file MATLAB.BAT in the MATLAB subdirectory. If you are using a menu system and MATLAB is one of your choices, just choose it. Otherwise, go to the MATLAB suiUirectory and type MATLAB.

You may be able to usea more powerful implementation of MATLAB if you have an 80286 or 80386 machine. AT-MATLAB runs on an 80286 with at least 1 Mbyte of extended memory. AT-MATLAB is distributed with PC-MATLAB. 386-MATLAB, a special version for 80386 or 80486 machines with virtual memory support and no limits on variable size, is sold separately.

When you run MATLAB, you should see the prompt . The program interpreter is waiting for you to enter instructions. Some MATLAB instructions, such as plot, are graphics-type instructions which plot results and data. Execution of one of these graphics instructions puts the PC screen into the *graphics mode*, which displays the resulting plot. No instructions can be executed in the graphics mode other than a screen-dump function. Striking any other key will return the PC to the *command mode*, but the graphics are temporarily stored (like variables) and can be recalled by the **shg** (show graphics) instruction. If you wish, you may run some of the demonstration programs now by entering demo and following the on-screen instructions.

An Introduction to MATLAB: Interactive Mode

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

In command mode, MATLAB displays a prompt and waits for your input. You may type any legal mathematical expression for immediate evaluation. Try the following three examples (press “enter” or “return” at the end of each line):

```
≫ 2+2
≫ 5^2
≫ 2*sin(pi/4)
```

The variable `pi = 3.14` is built into MATLAB, as are the `sin` function and hundreds of other functions. When you entered each of the preceding lines, MATLAB stored the results in a variable called `ans` for *answer*. The value of `ans` was then displayed. The last line should have produced the square root of 2. We can manipulate `ans` to find out

```
≫ ans^2
```

The new answer is very close to 2, as expected. Let's see what the roundoff error is:

```
≫ ans-2
```

An Introduction to MATLAB: Variables

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

Any result you wish to keep for a while may be assigned to a variable other than `ans`:

```
≫ x = pi/7
≫ cos(x)
≫ y = sin(x)^2+cos(x)^2;
≫ y
```

A semicolon (;) at the end of the line suppresses printing of the result, as when we calculated y in the next-to-last line just shown. This feature is especially useful when writing MATLAB programs where intermediate results are not of interest and when working with large matrices.

MATLAB supports the dynamic creation of variables. You can create your own variables by just assigning a value to a variable. For example, type `x = 3.5+4.2`. Then the real variable `x` contains the value `7.7`. Variable names must start with an alphabetical character and be less than nineteen characters long. If you type `x = -3*4.0`, the content `7.7` is replaced by the value `-12`. Some commands allow you to keep track of all the variables that you have already created in your session. Type `who` or `whos` to get the list and names of the variables currently in memory (`whos` gives more information than `who`). To clear all the variables, type in `clear`. To clear a single variable (or several) from the list, follow the command `clear` by the name of the variable you want to delete or by a list of variable names separated by spaces. Try it now.

MATLAB is case sensitive. In other words, x and X are two different variables. You can control the case sensitivity of MATLAB by entering the command `casesen`, which toggles the sensitivity. The command `casesen` on enforces case sensitivity, and `casesen off` cancels it.

If one line is not enough to enter your command, then finish the first line with two dots $(\ldots)$ and continue on the next line. You can enter more than one command per line by separating them with commas if you want the result displayed or with semicolons if you do not want the result displayed. For example, type

```
≫ theta = pi/7; x = cos(theta); y = sin(theta);
≫ x,y
```

to first compute `theta`,`cos(theta)`, and `sin(theta)` and then to print x and y.

An Introduction to MATLAB: Complex Variables

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

The number $\sqrt{-1}$ is predefined in MATLAB and stored in the two variable locations denoted by `i` and `j`. This double definition comes from the preference of mathematicians for using i and the preference of engineers for using j (with i denoting electrical current). `i` and `j` are variables, and their contents may be changed. If you type `j = 5`, then this is the value for `j` and `j` no longer contains $\sqrt{-1}$. Type in `j = sqrt(-1)` to restore the original value. Note the way a complex variable is displayed. If you type `i`, you should get the answer

```
i =
     0+1.0000i.
```

The same value will be displayed for `j`. Try it. Using `j`, you can now enter complex variables. For example, enter `z1 = 1+2*j` and `z2 = 2+1.5*j`. As `j` is a variable, you have to use the multiplication sign `*`. Otherwise, you will get an error message. MATLAB does not differentiate (except in storage) between a real and a complex variable. Therefore variables may be added, subtracted, multiplied, or even divided. For example, type in `x = 2, z = 4.5*j, and z/x`. The real and imaginary parts of `z` are both divided by `x`. MATLAB just treats the real variable `x` as a complex variable with a zero imaginary part. A complex variable that happens to have a zero imaginary part is treated like a real variable. Subtract `2*j` from `z1` and display the result.

MATLAB contains several built-in functions to manipulate complex numbers. For example, `real (z)` extracts the real part of the complex number `z`. Type

```
≫ z = 2+1.5*j, real(z)
```

to get the result

```
z =
   2.000+1.500i

ans =
   2
```

Similarly, `imag(z)` extracts the imaginary part of the complex number `z`. The functions `abs(z)` and `angle(z)` compute the absolute value (magnitude) of the complex number `z` and its angle (in radians). For example, type

```
≫ z = 2+2*j;
≫ r = abs(z)
≫ theta = angle(z)
≫ z = r*exp(j*theta)
```

The last command shows how to get back the original complex number from its magnitude and angle. This is clarified in Chapter 1: Complex Numbers.

Another useful function, `conj (z)`, returns the complex conjugate of the complex number `z`. If `z = x+j*y` where `x` and `y` are real, then `conj (z)` is equal to `x-j*y`. Verify this for several complex numbers by using the function `conj (z)`.

An Introduction to MATLAB: Vectors and Matrices

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

As its name indicates, MATLAB is especially designed to handle matrices. The simplest way to enter a matrix is to use an explicit list. In the list, the elements are separated by blanks or commas, and the semicolon (;) is used to indicate the end of a row. The list is surrounded by square brackets [ ]. For example, the statement

```
≫ A = [1 2 3;4 5 6;7 8 9]
```

results in the output

```
A =
   1 2 3
   4 5 6
   7 8 9
```

The variable A is a matrix of size 3×3. Matrix elements can be any MATLAB expression. For example, the command

```
≫ x = [-1.3 sqrt(3) (1+2+3)*4/5]
```

results in the matrix

```
x =
   -1.3000  1.7321  4.8000
```

We call a matrix with just one row or one column a vector, and a 1×1 matrix is a scalar. Individual matrix elements can be referenced with indices that are placed inside parentheses. Type `x(5) = abs(x(1))` to produce the new vector

```
x =
   -1.3000   1.7321   4.8000   0.000   1.3000
```

Note that the size of `x` has been automatically adjusted to accommodate the new element and that elements not referenced are set equal to 0 (here $\mathrm{x}(4)$). New rows or columns can be added very easily. Try typing `r = [10 11 12],A = [A;r]`. Dimensions in the command must coincide. Try `r = [13 14],A = [A;r]`.

The command `size(A)` gives the number of rows and the number of columns of `A`. The output from `size(A)` is itself a matrix of size 1×2. These numbers can be stored if necessary by the command `[m n] = size(A)`. In our previous example, `A = [A;r]` is a 4×3 matrix, so the variable `m` will contain the number 4 and `n` will contain the number 3. A vector is a matrix for which either `m` or `n` is equal to 1. If `m` is equal to 1, the matrix is a row vector; if `n` is equal to 1, the matrix is a column vector. Matrices and vectors may contain complex numbers. For example, the statement

```
≫ A = [1 2;3 4]+j*[5 6;7 8]
```

and the statement

```
≫ A = [1+5*j 2+6*j;3+7*j 4+8*j]
```

are equivalent, and they both produce the matrix

```
A =
   1.0000+5.0000i   2.0000+6.0000i
   3.0000+7.0000i   4.0000+8.0000i
```

Note that blanks must be avoided in the second expression for `A`. Try typing

```
≫A = [1 + 5*j 2 + 6*j 2 + 6*j;3 +7*j 4 + 8*j]
```

What is the size of `A` now?

MATLAB has several built-in functions to manipulate matrices. The special character, ', for prime denotes the transpose of a matrix. The statement `A = [ 1 2 3;4 5 6;7 8 9]'` produces the matrix

```
A =
   1 4 7
   2 5 8
   3 6 9
```

The rows of `A'` are the column of `A`, and vice versa. If `A` is a complex matrix, then `A` is its complex conjugate transpose or hermitian transpose. For an "unconjugate" transpose, use the two-character operator dot-prime (.'). Matrix and vector variables can be added, subtracted, and multiplied as regular variables if the sizes match. Only matrices of the same size can be added or subtracted. There is, however, an easy way to add or subtract a common scalar from each element of a matrix. For example, `x = [1 2 3 4],x = x-1` produces the output

```
x =
   1 2 3 4

x =
   0 1 2 3
```

As discussed in the chapter on linear algebra, multiplication of two matrices is only valid if the inner sizes of the matrices are equal. In other words, `A*B` is valid if the second size of `A` (number of columns) is the same as the first size of `B` (number of rows). Let $\mathrm{a}_{i,j}$ represent the element of `A` in the ith row and the jth column. Then the matrix `A*B` consists of elements
Equation:

$$(\mathrm{AB})_{i,j} = \sum_{k=1}^{n} \mathrm{a}_{i,k} b_{k,j}$$

where n is the number of columns of `A` and the number of rows of `B`. Try typing `A = [1 2 3;4 5 6];B = [7;8;9]; A*B`. You should get the result

```
ans =
    50
    112
```

The inner product between two column vectors `x` and `y` is the scalar defined as the product `x'*y` or equivalently as `y'*x` For example, `x = [1;2],y = [3;4],x'*y`, leads to the result

```
ans =
    11
```

Similarly, for row vectors the inner product is defined as `x*y'`. The Euclidean norm of a vector is defined as the square root of the inner product between a vector and itself. Try to compute the norm of the vector `[1 2 3 4]`. You should get `5.4772`. The outer product of two column (row) vectors is the matrix `x*y' (x'*y)`.

Any scalar can multiply or be multiplied by a matrix. The multiplication is then performed element by element. Try `A = [1 2 3;4 5 6;7 8 9];A*2`. You should get

```
ans =
    2   4   6
    8  10  12
   14  26  28
```

Verify that `2*A` gives the same result.

The inverse of a matrix is computed by using the function `inv(A)` and is only valid if `A` is square. If the matrix is singular, meaning that it has no inverse, a message will appear. Try typing `inv(A)`. You should get

```
Warning: Matrix is close to singular or badly scal
ed.
   Results may be inaccurate. RCOND=2.937385e-18

ans =
     1.0e+16*
     0.3152  -0.6304   0.3152
    -0.6304   1.2609  -0.6304
     0.3152  -0.6304   0.3152
```

The inverse of a matrix may be used to solve a linear system of equations. For example, to solve the system

you could type `A = [1 2 3;1 -2 4;0 -2 1]; b = [2;7;3]; inv(A)*b` and get

```
ans =
    1
   -1
    1
```

Equation:

$$\begin{matrix} 1 & 2 & 3 \\ 1 & -2 & 4 \\ 0 & -2 & 1 \end{matrix} \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} = \begin{matrix} 2 \\ 7 \\ 3 \end{matrix},$$

Check to see that this is the correct answer by typing `A*[1;-1;1]`. What do you see?

MATLAB offers another way to solve linear systems, based on Gauss elimination, that is faster than computing the inverse. The syntax is `A\b` and is valid whenever `A` has the same number of rows as `b`. Try it.

The "Dot" Operator. Sometimes you may want to perform an operation element by element. In MATLAB, these element-by-element operations are

called array operations. Of course, matrix addition and subtraction are already element-by-element operations. The operation `A. *B` denotes the multiplication, element by element, of the matrices `A` and `B`. Make two 3 $\times$3 matrices, `A` and `B`, and try

```
≫ A*B
≫ A.*B
```

Suppose we want to find the square of each number in `A`. The proper way to specify this calculation is

```
≫ A_squared=A.^2
```

where the period (dot) indicates an "array operation" to be performed on each element of `A`. Without the dot, `A` is multiplied by `A` according to the rules of matrix multiplication described in Chapter 4, giving a totally different result:

```
≫ A^2
```

Subtleties. Because MATLAB can do so many different mathematical functions with just a few keystrokes, there are times when a very slight change in what you type will lead to a different result. Using the matrix `A` entered earlier, type the following two lines:

```
≫ 2.^A
≫ 2 .^A                          %with a space after
the 2.
```

In the first case, the dot is "absorbed" by the 2 as a decimal point and the ^ is taken as a matrix exponential. But, when the dot is separated from the 2 by a space, it becomes part of the operator (. ^) and specifies that 2 should be raised to the power of each element in A. The point is, you should be very careful to type what you mean in an unambiguous way until you are familiar enough with MATLAB to know how the subtle situations wiIl be interpreted. An unambiguous way of typing the preceding lines is

```
≫ (2.)^A                    %for matrix
exponential
≫ (2).^A                    %for array
exponential.
```

An Introduction to MATLAB: The Colon

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

You can use the colon several ways in MATLAB (see help :). Its basic meaning is a vector of sequential values. For example, type

```
≫ x = 3:9
```

to get

```
x =
   3 4 5 6 7 8 9
```

For increments other than 1, use statements like

```
≫ x = 1:0.5:4
≫ x = 6:-1:0
```

Most MATLAB functions will accept vector inputs and produce vector outputs. The statement

```
≫ y = sqrt(1:10)
```

builds a vector of integers from 1 to 10 and takes the square root of each of those numbers. Try it.

Now for another subtlety–what is the effect of each of the following statements and why?

```
≫ 1+1:5
≫ 1+(1:5)
```

Appending to a Matrix or Vector. A matrix or vector can be enlarged in size by appending new values to the old values. Let `x=[1 3 5]`:
Equation:

$$\begin{aligned} \mathrm{x} &= [\mathrm{x}6810] \Rightarrow \mathrm{x} = [1356810] \\ \mathrm{y} &= [\mathrm{x}; 1:6] \Rightarrow \mathrm{y} = \begin{bmatrix} 1 & 3 & 5 & 6 & 8 & 10 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}. \end{aligned}$$

An Introduction to MATLAB: Graphics

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

In "Complex Numbers", you learn that complex variables can be represented as points in the plane. MATLAB makes it easy for you to plot complex variables in a graph. Type `z1 = 1+.5*j;plot(z1,'o')`. The graph window should be activated and the point `z1` displayed by a 'o'. You must specify the symbol for display, and the authorized symbols for point display are `'.', 'o', '+',` and `'*'`. When you are displaying a curve (to come later), no type is necessary. MATLAB automatically adjusts the scale on a graph to accommodate the value of the point being plotted. In this case, the range is $[0, 2]$ for the real part and $[0, 1]$ for the imaginary part.

Let's now plot a second complex number by typing `z2 = 2+1.5*j; plot(za,'o')`. Note that the second `plot` command erases the first `plot` and changes the scaling to $[0, 4]$ and $[0, 3]$. Sometimes you may want to have the points plotted on the same graph. To achieve this, you have to use the command hold on after the first plot. Try the following:

```
≫ plot(z2,'o')
≫ hold on
≫ plot(z1,'o')
≫ hold off
```

The advantage in using the `hold` command is that there is no limit to the number of plot commands you can type before the hold is turned off, and these plots may involve the same variable plotted over a range of values.

You can also use different point displays. A disadvantage of the `hold` command is that the scaling is enforced by the first plot and is not adjusted for subsequent plots. This is why we plot the point `z2` first. Try reversing the order of the plots and see what happens. This means that points outside the scaling will not be displayed. The command `hold off` permits erasing the current graph for the next `plot` command.

You can freeze the scaling of the graph by using the command `axis`. MATLAB gives you the message

```
Axis scales frozen
ans =
   0 4 0 3
```

This freezes the current axis scaling for subsequent plots. Similarly, if you type axis a second time, MATLAB resumes the automatic scaling feature and prints the message

```
Axis scales auto-ranged
ans =
   0 4 0 3
```

The axis scaling can also be manually enforced by using the command

```
≫ axis([xmin xmax ymin ymax])
```

where `(xmin, ymin)` is the lower left corner and `(xmax, ymax)` is the upper right corner of the graph. This scaling remains in effect until the next `axis` command is entered (with or without arguments).

Another way to plot several complex numbers on the same graph is to display them as a curve. For this purpose, you have to store the numbers in a vector. For example, type `z1 = z1,z(2) = z2,plot(z)`. Note that the two points are at the two extremes of the line plotted on the graph. If you specify a symbol, then no line is drawn, just the extreme points. Try `plot(z,'o')` .

If you examine your current graph carefully, you will notice that the unit lengths on the *x* and *y* axes are not quite the same. In fact, MATLAB adjusts the length of an axis to conform to the overall size of the graph window. What this means is that a 45° line will actually be displayed at an angle depending on the overall aspect ratio of the graph window. To ensure that the aspect ratio is equal to 1, you may enter the command `axis('square')`. MATLAB will then enforce an aspect ratio equal to 1, regardless of the aspect ratio for the outside graph window. This ensures that circles appear as circles and not as ellipses. MATLAB will make the square graph as large as possible to fit within the graph window. To go back to the default ratio, just type in `axis('normal')`.

To add labels to your graph, the functions `xlabel('text')`, `ylabel('text')`, and `title('text')` are useful and self-explanatory. The argument `text` contains a string of characters. Add the label `Real` on the horizontal axis and the label `Imaginary` on the vertical axis of your graph. The command `grid` draws a grid on your graph. The grid does not remain in effect for the next plots. Try it.

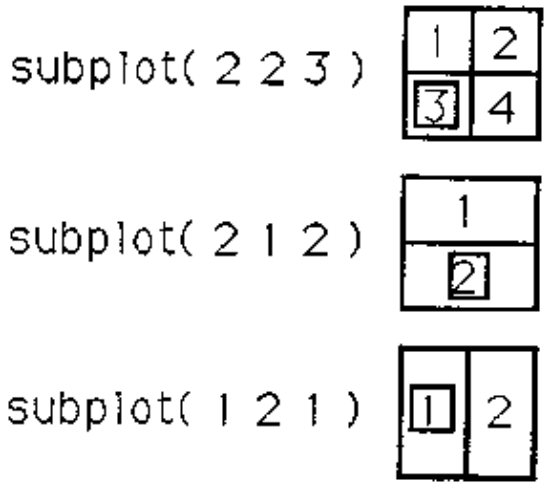
The plot Instruction. The plot instruction in MATLAB is very versatile. It can be used to plot several different types of data. Its syntax is `plot(x,y,' symbol and/or color')` or `plot(y,'symbol and/or color')`. The instruction will plot a vector of data versus another vector of data. The first vector is referenced to the horizontal axis and the second to the vertical axis. If only one vector is used, then it is plotted with reference to the vertical axis while the horizontal axis is automatically forced to be the index of the vector for the corresponding data point. The notation inside the apostrophes is optionally used to designate whether each element of the vector is to be plotted as a single point with a certain symbol or as a curve with a straight line drawn between each data value. The colors can also be specified. Possible symbols are `(*,o,+,.)`, and colors are `(r,g,b,w)` (red, green, blue, and white). Complex valued vectors are plotted by making the horizontal axis the real part of the vector and the vertical axis the imaginary part. Warning: A complex valued vector will automatically be plotted correctly on the complex plane (instead of real versus imaginary) *only* if every element of the vector is complex valued. Try `plot (x,y,'*')` , `plot (x,'*')`, `plot (y,'*')`, `plot`

`(y,x,'*')` and `plot (z,'*')`, `plot (Real (z), Imag (z),'*')` to clarify your understanding of plot. Use $\mathrm{x} = [1357], \mathrm{y} = [2468], \mathrm{z} = [1 + j, 2 + 2j, 3 + 3j]$.

We may summarize as follows:

`plot(x,'*r')`	(red star—points with the values of x on vertical and indicies on horizontal)
`plot(y)`	(line—connected curve of the value of y on vertical and the value of x on horizontal)
`plot(x,y,'og')`	(line—connected curve of the value of y on vertical and the value of

	x on horizontal)
`plot(x,y,'og')`	(circle—points of the value of y on vertical and the value of x on horizontal)
`plot(real(z),imag(z))`	(line—connectedplot of z on the complex plane)
`plot(real(z),imag(z),'+b')`	(blue plus—points of z on the complex plane).



Subplots. It is possible to split the graphics screen up into several separate smaller graphs rather than just one large graph. As many as four subplots can be created. The MATLAB instruction `subplot (mnp)` signifies which of the smaller graphs is to be accessed with the next plot statement. The mnp argument consists of three digits. The `m` and `n` are the numbers of rows (`m`) and columns (`n`) into which the screen should be divided. The `p` designates which of the matrix elements is to be used. For example,

Help and Demos. MATLAB has on-line help and a collection of demonstrations. For a list of available functions, type

```
≫ help
```

For help on a specific function, `sin` for example, type

```
≫ help sin
```

To learn how to use colon (;, a very important and versatile character) in MATLAB, type

```
≫ help :
```

The demos will also help you become more familiar with MATLAB and its capabilities. To run them, type

```
≫ demo
```

An Introduction to MATLAB: Editing Files and Creating Functions (Macintosh)

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

If you quit MATLAB now, all the commands you have typed will be lost. This is where the *Edit* window is useful. If you choose *new* from the *File* menu, a new window appears with the title *Edit 1*: *Untitled*. In this window, you should type in all the commands you would like MATLAB to execute at once. When you are finished typing, you may save the file by choosing *save* or *save as* in the file menu and by entering a name for the file. If the edit window is active (that is, if it appears in front), then choose *save and go* from the file menu to save the file and execute it. If the command window is active, then you can execute the file by entering its name.

Editing Files. To test your understanding of file editing, enter the following commands in a file named myfile:

```
clear, clg                          %Clear variables a
nd graphics
j=sqrt(-1);                         %To be sure
z1=1+.5*j,z2=2+1.5*j                %Enter variables
z3=z1+z2,z4=z1*z2                   %Compute sum and p
roduct
axis([0 4 0 4]),                    %First plot
  axis('square'),plot(z1,'o')
hold on                             %Allow overplot
plot(z2,'o'),plot(z3,'+'),          %Other plots
```

```
    plot(z4,'*')
hold off
```

You do not have to type the % sign and the text that follows it. These are simply comments in a file. They are ignored by the MATLAB interpreter. You should, however, make a habit of adding comments (preceded by %) to your file if you want to be able to understand programs that have been written long ago.

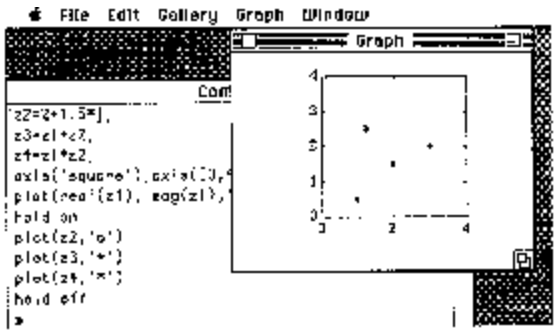
Do not forget to save your file. Such a file is called a *script file*. It contains MATLAB commands that could have been entered one by one in the command window. You have three ways to execute a script file:

i. with the *edit* window active, choose *save and go* from the *file* menu;
ii. with the *command* window active, enter the file name; or
iii. with the *command* window active, choose *run script...* from the *M-file* menu. In this case, a menu pops up to ask you which file you want to execute.

Try each of these three methods in order to get used to their differences. Figure 1 shows the plot that you should get.

Creating Functions. MATLAB puts many commands at your disposal, and you just have to enter their names (with or without arguments) to execute them. Some commands are built in to MATLAB, and others are contained in files to which you have access (not to modify them, but to see how things are done). You can try to display the contents of a command file in the command window by entering the command type filename. Enter type plot. You should get the message

```
??? Built-in function.
```



A Typical Graph (©Apple Computer, Inc., used with permission.)

This means that the command `plot` is a build-in function and thus cannot be typed by the user (it is compiled with the program). Similarly, try to type the function file containing the command `sinh` by entering the command `type sinh`. You should get

```
function y=sinh(x)
%SINH SINH(X) is the hyperbolic sine of the elemen
ts of X.
y=(exp(x)-exp(-x))/2;
```

This is a typical example of a *function file*. It differs from a script file by the fact that the first line contains the word `function`. What it does is make the command `y = sinh(x)` equivalent to the command `y=(exp(x)-exp(-x))/2` . The variable `x` is the input argument (there could be more than one), and *y* is the output argument (there also could be more than one). The second line contains comments about the function and its arguments. They can spread to several lines. Display them by typing the command `help sinh`. The third line contains the actual function commands (there could also be several lines).

As an exercise, enter the function `perp(x)`:

```
function y=perp(x)
%PERP PERP(X) is a complex value perpendicular to
X.
j=sqrt(-1);
y=j*real(x)-imag(x);
```

Evaluate `perp` on various complex numbers. Replace the last line by `y = x*j`;. Do you get the same result? Why?

Note that local variables are just that–*local*. Only the input and out- put arguments are kept in memory after the function is called and executed. For more on functions and multiple arguments, see the MATLAB manual. In the previous example, the variable j is local. If, before using the function `perp`, you use the same variable j, its contents will not be affected by the command `perp(z)`. Verify this.

Normally, while a file (script or function) is executed, the commands are not displayed on the command window. Just the results are displayed. The command `echo` allows you to view all the instructions. This is useful for debugging and demonstrations. It is turned off by typing `echo off`.

An Introduction to MATLAB: Editing Files and Creating Functions (PC)

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

You should write a MATLAB program whenever you anticipate exe- cuting some sequence of statements several times or again in a later session. On an IBM PC, you may use any text editor to write a program, as long as the file can be saved in ASCII format without the control codes used by most word processors. Applicable text editors include Edix, Wordstar, XTree Pro, and Turbo Pascal's editor.

Editing Files. If you have enough memory, you can run your editor without leaving MATLAB by using the exclamation point (!), like this:

```
≫ !EDIX
```

The exclamation point may be used to execute any DOS command or program from MATLAB. When the command or program finishes, your MATLAB variables are just as you left them. Use your editor to write program lines just as you would type them in MATLAB's command mode. Then save the file with extension `.m` in the directory where you will run MATLAB. Such MATLAB programs are called m-files. You may run your m-file by typing the file name (without the `.m` extension) at the MATLAB command prompt (≫).

Script Files. There are two kinds of m-files, called *script files* and *functions*. Running a script file is exactly like typing the commands it contains at the ≫ prompt. Your m-file will automatically be a script file unless you specify otherwise, as described later. Practice by entering, saving, and running `plotsin.m` as listed next:

```
t = -6:.2:6;
y = sin(t);
plot(t,y)
title('SINE')
pause
grid
xlabel('t')
ylabel('sin(t)')
```

When the pause is executed, you will need to press a key to go on. If you type whos after running `plotsin`, you will see that the variables `t` and `y` remain in memory. Comments are important to a script file. They are marked with the symbol %. Anything following this symbol on a line is assumed to be a comment and is ignored by the MATLAB program interpreter.

Functions. Functions differ from script files in that they have designated input and output variables. Any other variables used within a function are local variables, which do not remain after the function terminates and which have no effect on variables outside the function. Many of the functions supplied with MATLAB are actually m-files. A good example is `triu.m`:

```
≫ type triu
```

The word function at the beginning of the file makes it a function rather than a script file. The function name in this line must match the file name. The input variables of `triu` are `x` and `k`, meaning that the first input argument will be referred to as `x` and the second as `k` within the function. Likewise, the function line designates `y` as the output. There is nothing special about the variable names `x`,`k`, and `y` when the function is used. It is only that whatever inputs and output you use will be referred to as `x`,`k`, and `y` inside the function. The variables `m`,`n`,`j`, and `i` are created temporarily when `triu` runs and disappear when it terminates. They are local variables and have no effect on variables with the same names outside the function. In contrast, a script file has no local variables and does no substitution of input and output variable names.

As an exercise, enter and save the function `perp.m`:

```
      function y=perp(x)
      % PERP(x) is a complex number perpendicular
to x.
      j = sqrt(-1);
      y = j*real(x)-imag(x);
```

Evaluate `perp` on various complex numbers. Replace the last line by `y = x*j;`. Do you get the same result? Why?

Printing Files and Graphics. To display an m-file on the screen, use the instruction `≫ type filename`. To make a copy at a printer, use the DOS command `≫ !print filename.m`. Graphics hardcopy is available through the commands meta and `gpp`. See the MATLAB manual for more information.

An Introduction to MATLAB: Loops and Control

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

MATLAB has control statements like those in most computer languages. We will only study the `for` loop here. See the MATLAB manual for details on `if` and `while` statements.

What for loops do is allow a statement or a group of statements to be repeated. For example,

```
for i = 1:n,x(i) = 0,end
```

assigns the value 0 to the first `n` elements of the array `x`. If `n` is less than 1, the instruction is still valid but nothing will be done. If `n` is not defined, then the following message will appear:

```
??? Undefined function or variable.
Symbol in question==>n
```

If `n` contains a real value, the integer part of `n` is used. If `n` is a complex number, the integer part of the real part is taken. (This should, however, be avoided.) If `x` is not declared earlier or is of smaller dimension than `n`, then extra memory is allocated.

The command end must follow the command for. If it is not present, MATLAB will wait for remaining statements until you type *the* command end, and nothing will be executed in the meantime.

More than one statement may be included in the loop. The statement `1:n` is the way to specify all the integer values between 1 and `n`. A step different than 1 may be specified. For example, the statement `for i=1:2:5,x(i)=1,end` is equivalent to `x(1)=1,x(5)=1`. Negative steps are also allowed, as in `i=n:-1:1`.

We may use a `for` loop to draw a circle of radius 1. Type

```
≫ j=sqrt(-1);
≫ n=360;
≫ for i=1:n,circle(1)=exp(2*j*i*pi/n);end;
≫plot(circle)
```

Note how easy it is to plot a curve. But also note how slowly the for loop is executed. MATLAB is not very good at executing things one by one. It prefers, by far, a vector-oriented statement. Using the range specification as in the `for` loop, it is possible to speed up the process by replacing the explicit `for` loop by an implicit `for` loop using the colon, like this:

```
≫ circle = exp((2*j*pi/n)*[1:n]);
```

Note how much faster this command is executed. In general, `for` loops should be avoided as much as possible. For example, the first `for` loop you typed could have been replaced by the command `x=zeros(1,n);`, which is much more efficient. The function `zeros` fills the variable with 0's to the specified size. Similarly, the function `ones` fills the variable with 1's. The size can also be determined by the size of the argument. If `A` is a matrix of size `m,n`, then the command `B=ones(a)` fills the matrix `B` with 1's and forces the matrix `B` to have exactly the same size as the matrix `A`.

Avoiding `for` Loops. Since `for` loops are very inefficient in MATLAB (they are computed sequentially, adding several more computations for every loop), it is preferable to use the matrix capabilities of MATLAB to replace `for` loops and speed up processing time.

i. Replace the for loop
`≫ for i = 1:10, ≫ x(i) = i; ≫ end;`

with

```
≫ x = 1:10
```

to get

```
x = [1 2 3 4 5 6 7 8 9 10]
```

ii. Replace the `for` loop

```
≫ z = something; ≫ for i = 1:10, ≫ x(i) =
z*i; ≫ end;
```

with

```
≫ z = something ≫ x = z*(1:10);
```

to get

```
x = [z 2*z 3*z ... 10*z]
```

iii. Replace the `for` loop

```
≫ z = something; ≫ x(1) = z; ≫ for i = 2:10.
≫ x(i) = z*x(i-1); ≫ end;
```

with

```
≫ z = something; ≫ x = z.^(1:10);
```

to get

```
x = [z z**2 z**3 ... z**10]
```

iv. Replace the `for` loop

```
≫ for i = 0:2:100, ≫ x(i) =
1.0*exp(j*2*pi*i/100); ≫ end;
```

with

```
≫ x = 1.0*exp(j*2*pi*(0:2:100)/100);
```

to get

```
x = [exp(0) exp(j*4*pi/100) ...
exp(j*200*pi/100)]
```

The Edix Editor

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

On a PC, creating and editing m-files is done outside of MATLAB. The Edix editor is one of the visual types of editors. It can be accessed either from DOS by using `> edix filename.m` or from MATLAB by using `≫ !edix filename.m`. The latter will return you back to MATLAB after you quit the editor. MATLAB script files must end with a `.m` extension. They are also called m-files. Once in the Edix editor, a help list of the editor's commands can be displayed on the screen by typing “Alt-h.” (Note: The “Alt” key must be held down while the "h" key is depressed.) Pressing the “space” key will get you out of the help system.

A new file is created by calling the Edix editor with a file name that has not been previously used. If the file name already exists on disc, then that file is retrieved and displayed on the screen and is ready to be edited. Since Edix is a visual editor, changes are made by moving the cursor to wherever something needs to be typed in or deleted. Once changes have been made to a file, then that file needs to be saved (written to disc) by typing “Alt-w.” When all editing is finished, you may exit from the Edix editor by typing “Alt-x.” Following are some useful Edix editor commands:

Alt-h(F7)	help-list of all the Edix commands;

Alt-w	saves (writes) the file to disc;
Alt-x	exits the Edix editor;
Alt-d	deletes the current line;
Alt-m (F8)	the first time used, marks the beginning of a block of lines to be edited; the second time, marks the end of a block of lines to be edited and highlights this block in white; the third time, "un-marks" the highlighted block;
Alt-c	makes a copy of the highlighted block at the current cursor location (can be repeated);
Alt-m	(moves the highlighted block to the current cursor location (can be repeated);
(F5)	put the cursor at the beginning of the line; and
(F6)	puts the cursor at the end of the line.

Useful Mathematical Identities

Note: This module is part of the collection, *A First Course in Electrical and Computer Engineering*. The LaTeX source files for this collection were created using an optical character recognition technology, and because of this process there may be more errors than usual. Please contact us if you discover any errors.

$e^{j\theta}$, cos θ, and sin θ

$$e^{j\theta} = \lim_{n\to\infty}\left(1+j\frac{\theta}{n}\right)^n = \sum_{n=0}^{\infty}\frac{1}{n!}(j\theta)^n = \cos\theta + j\sin\theta$$

$$\cos\theta = \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n)!}\theta^{2n}; \sin\theta = \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}\theta^{2n+1}$$

$$\cos\theta = \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n)!}\theta^{2n}; \sin\theta = \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}\theta^{2n+1}$$

Trigonometric Identities

Equation:

$$\sin^2\theta + \cos^2\theta = 1$$

Equation:

$$\sin(\theta+\varphi) = \sin\theta\cos\varphi + \cos\theta\sin\varphi$$

Equation:

$$\cos(\theta+\varphi) = \cos\theta\cos\varphi - \sin\theta\sin\varphi$$

Equation:

$$\sin(\theta-\varphi) = \sin\theta\cos\varphi - \cos\theta\sin\varphi$$

Equation:

$$\cos\left(\theta - \varphi\right) = \cos\theta\cos\varphi + \sin\theta\sin\varphi$$

Euler's Equations

Equation:

$$e^{j\theta} = \cos\theta + j\sin\theta$$

Equation:

$$\sin\theta = \frac{e^{j\theta} - e^{-j\theta}}{2j}$$

Equation:

$$\cos\theta = \frac{e^{j\theta} + e^{-j\theta}}{2}$$

De Moivre's Identity

Equation:

$$\left(\cos\theta + j\sin\theta\right)^n = \cos n\theta + j\sin n\theta$$

Binomial Expansion

Equation:

$$\left(x + y\right)^N = \sum_{n=0}^{N} \binom{N}{n} x^n y^{N-n} \; ; \quad \binom{N}{n} = \frac{N!}{(N-n)!n!}$$

Equation:

$$2^N = \sum_{n=0}^{N} \binom{N}{n}$$

Geometric Sums

Equation:

$$\sum_{k=0}^{\infty} az^k = \frac{a}{1-z} \quad |z| < 1$$

Equation:

$$\sum_{k=0}^{N-1} az^k = \frac{a\left(1-z^N\right)}{1-z} \quad z \neq 1$$

Taylor's Series

Equation:

$$f\left(x\right) = \sum_{k=0}^{\infty} f^{\left(k\right)}\left(a\right)\frac{\left(x-a\right)^k}{k!}$$

Equation:

$$\left(\text{Maclaurin's Series if } \ a=0\right)$$